Universidad Complutense de Madrid

Facultad de Informática U.C.M

OPTIMIZACIÓN DE RUTAS DE TRANSPORTE

Proyecto de Sistemas Informáticos

Realizado por: Andrés Aguado Aranda, Javier Jiménez de Vega

Dirigido por: José Jaime Ruz Ortiz

Curso 2012/2013

RESUMEN

En este proyecto hemos desarrollado una aplicación de escritorio que genera una ruta de transporte de pasajeros, determinando la mejor forma de realizar el recorrido y tratando de reducir los costes de la empresa de transporte al mínimo. La aplicación recibe los datos de los domicilios de los pasajeros, los agrupa, en la medida de lo posible, en función de ciertos parámetros en paradas mediante algoritmos de Clustering y determina la forma óptima de recogerles y transportarles mediante un algoritmo genético. Todo esto está implementado ayudándonos del proveedor de mapas online, Google Maps, del que se toman y reflejan datos de mapas reales.

En las siguientes secciones procederemos a explicar cómo hemos obtenido la ruta óptima para el transporte de los pasajeros e introduciremos los conceptos clave en los que hemos basado el proyecto, necesarios para comprender el funcionamiento de la aplicación. A continuación, detallaremos la implementación de la aplicación y acabaremos con las conclusiones que hemos extraído, incluyendo posibles ampliaciones del proyecto.

Palabras clave: Optimización, Clustering, Algoritmos Genéticos, Servicios Web, Web Mapping.

ABSTRACT

In this project we have developed a desktop application that manages a passenger route, determining the best way to make the journey and trying to reduce the cost of the carrier, to the minimum. The application receives data from the homes of the passengers, groups them by clustering algorithms in stops, if possible, and determines the optimal way to pick them up and transport them using a genetic algorithm. All of this is implemented using online maps, like Google Maps, which we take online maps.

In the following sections we explain what is and how a route works. Then we introduce the key concepts on which we based the project, necessary to understand the functioning of the application. Finally, we will explain deeply the application development, ending with the conclusions we have drawn, including possible extensions of the project.

TABLA DE CONTENIDO

CAPÍTULO I: INTRODUCCIÓN

INTRODUCCIÓN

En este primer apartado de la memoria se realiza una descripción general del tema que abordará el proyecto y los objetivos a cumplir.

PRESENTACIÓN DEL TEMA

El auge de la informática ha mejorado la calidad de vida de sus usuarios, tanto a gran escala, como el caso de las grandes empresas, como a nivel de usuario particular. Una de las principales tareas para la que se usan los computadores es la optimización de recursos para aumentar la eficiencia y el rendimiento de las correspondientes tareas, traduciéndose en mejoras económicas a medio plazo. Concretamente, la optimización de rutas en medios de transporte es uno de los grandes ahorros que se han conseguido gracias a la utilización de estas técnicas.

El diseño y optimización de rutas es tremendamente ventajoso en cualquier ámbito y situación para cualquier tipo de usuario, en especial para las empresas de transporte cuyas pérdidas y ganancias se basan en la distribución óptima tanto del tiempo, como del combustible, que están directamente relacionadas con la distancia recorrida. Esto es aplicable a cualquier tipo de vehículo aéreo, terrestre o marítimo.

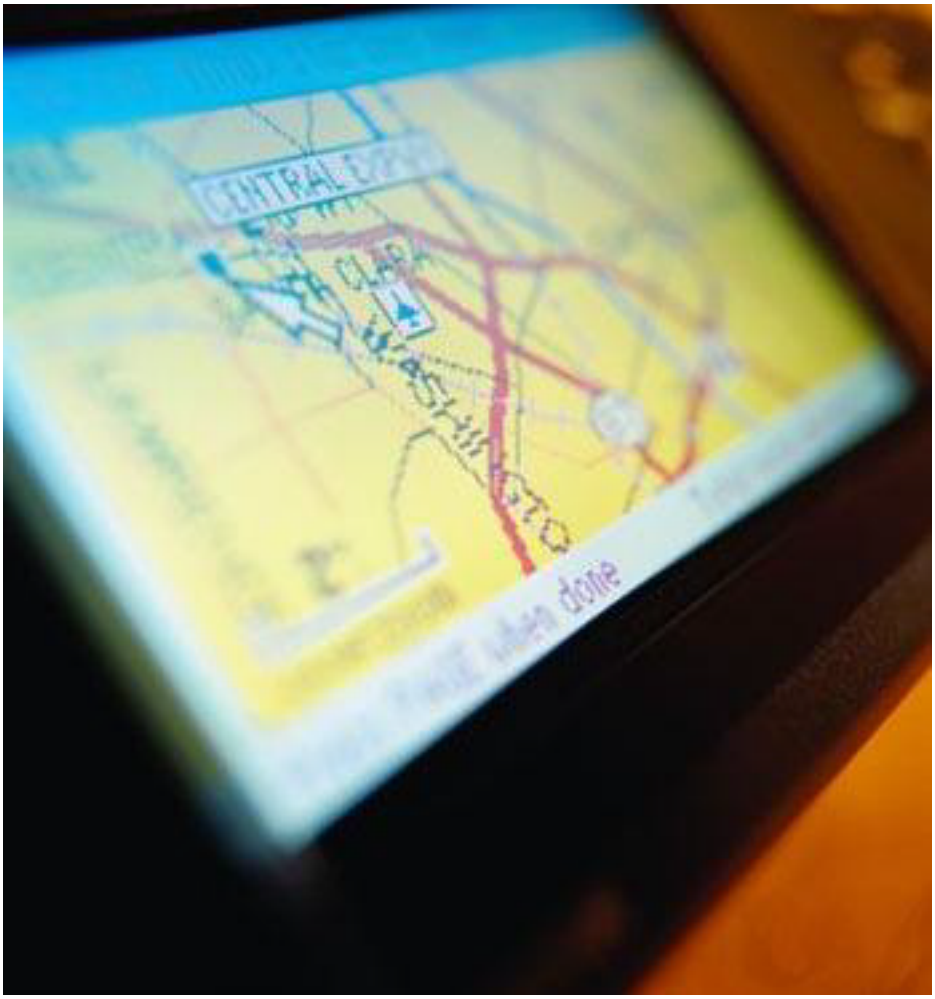

Ilustración 1 - GPS

Una de las funciones que más ha evolucionado en los últimos años en las organizaciones es la de la distribución. Sin embargo, esta evolución ha derivado inexorablemente en un incremento de la complejidad de las operaciones de transporte, lo que unido a diversos factores, tales como la necesidad de reducir los costes de producción, el constante incremento de los precios del transporte o el aumento de los niveles de exigencia en las relaciones cliente-proveedor, han situado a la gestión logística como un elemento clave dentro de la estrategia de las empresas.

Un gestor de rutas es una aplicación informática que a partir de un conjunto de puntos, entre ellos un punto de destino final prefijado, selecciona un punto inicial y una ruta que recorra los restantes puntos intermedios de manera óptima, es decir, minimizando el tiempo y la distancia de dicha ruta.

Un factor fundamental para la optimización de rutas son los algoritmos de agrupamiento o Clustering, que posteriormente explicaremos su funcionamiento y aplicación a este proyecto. El análisis de clústeres es una colección de métodos

estadísticos que permiten agrupar casos sobre los cuales se miden diferentes variables o características. De este modo podremos formar grupos de puntos próximos y fijar una parada intermedia, reduciendo así el tiempo y el coste de combustible de la ruta.

OBJETIVO Y ALCANCE

El proyecto consiste en la elaboración de una aplicación de gestión de rutas que permita elaborar, mediante algoritmos de Clustering y genéticos, un recorrido óptimo de recogida de trabajadores de una empresa.

Como datos de entrada de la aplicación tenemos las ubicaciones de las viviendas de los pasajeros de la ruta. Los empleados rellenarán una plantilla con la información correspondiente y se la enviarán al administrador. Éste colocará los archivos en el directorio oportuno para que la aplicación incorpore a la base de datos el contenido de estas plantillas.

El programa agrupará las viviendas en un conjunto de paradas intermedias de tal manera que no se supere una distancia máxima entre las viviendas de la misma parada. Para ello se utilizarán algoritmos de Clustering. Después recorrerá dichas paradas de forma óptima. El orden será calculado mediante un algoritmo genético. Al acabar de procesar toda la información, devolverá un mapa donde se mostrará la ruta a seguir por el conductor del transporte contratado por la empresa.

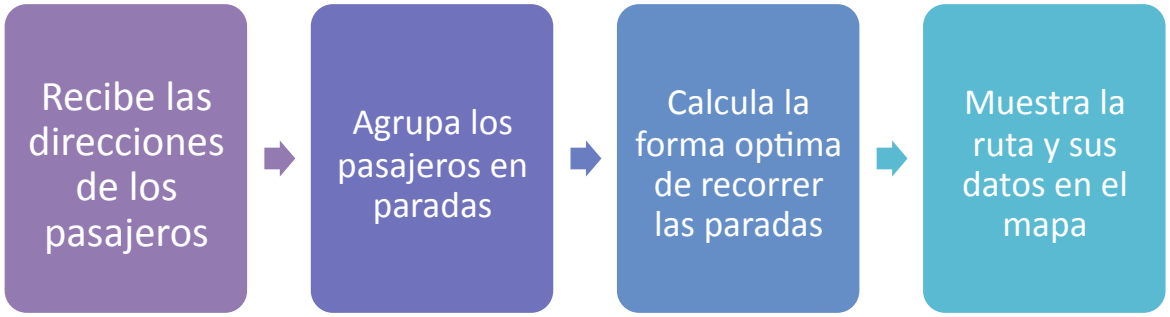

Ilustración 2 - Esquema circuito de la aplicación

Este es un ejemplo del resultado que mostrará la aplicación:

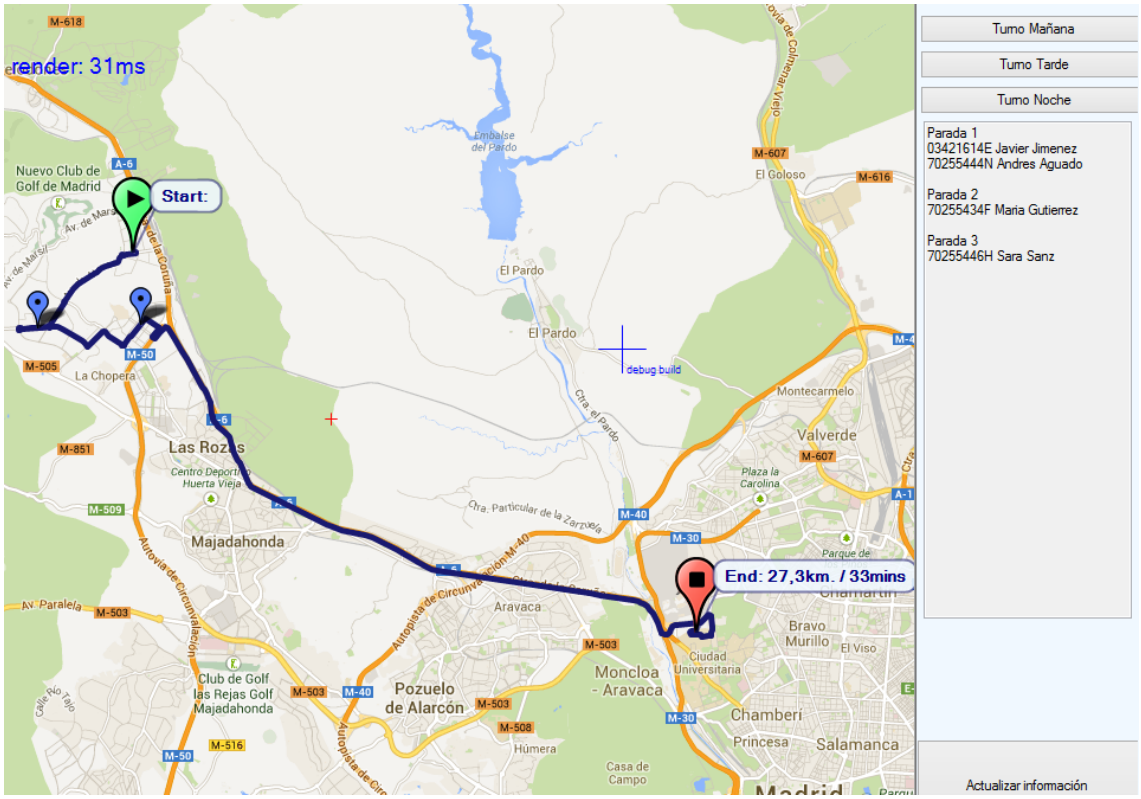

Ilustración 3 - Ejemplo muestra de una ruta

RELACIÓN CON LOS ESTUDIOS

Para la realización de este proyecto hemos utilizado diversos conceptos aprendidos a lo largo de las diferentes asignaturas cursadas durante la carrera. También hemos encontrado útiles conceptos externos a la misma, adquiridos a través de la experiencia profesional y formación complementaria a la Ingeniería Informática, como cursos, seminarios, etc...

Las asignaturas estudiadas en la facultad que nos han resultado más útiles para la planificación y el desarrollo de este proyecto han sido las siguientes:

Ingeniería del Software

Esta asignatura ha resultado fundamental en la realización del proyecto durante todas sus fases: elaboración de requisitos, diseño, implementación y gestión. Gracias a los conocimientos de elaboración de documentos y desarrollo estructurado, hemos sido capaces de realizar el proyecto de una forma más eficiente.

Programación Evolutiva

Debido a que la forma de optimizar las rutas se ha realizado mediante algoritmos genéticos, el estudio de esta asignatura ha sido clave.

Metodología y Tecnología de la Programación

Esta asignatura nos ha ayudado a mejorar la eficiencia de los algoritmos comunes utilizados para el proyecto y para el diseño del algoritmo de Clustering que permite el agrupamiento de los pasajeros en paradas cercanas. Además, hemos utilizado el algoritmo del *viajante* para comparar resultados con nuestro algoritmo genético.

CAPÍTULO II: CONCEPTOS Y ESTADO DEL ARTE

INTRODUCCIÓN AL CAPÍTULO

En este capítulo se hará una breve descripción de los conceptos que se emplean en este proyecto para el correcto entendimiento de la aplicación. Explicaremos el concepto de optimización de rutas de transporte y sus diferentes puntos a considerar.

Posteriormente explicaremos cómo hemos resuelto el problema de agrupación mediante Clustering, donde definiremos en qué consiste este concepto y cómo lo hemos aplicado.

Después explicaremos en profundidad qué es un Algoritmo Genético y su papel en la aplicación, ya que es la piedra angular de la optimación en nuestro proyecto. Para ello comentaremos los diferentes conceptos que se usan en la aplicación y cómo los hemos utilizado para resolver el problema de la optimización de rutas reales.

Finalmente abordaremos el tema del "Web Mapping", o "cartografía en la web", de la que nos hemos servido para representar los mapas y obtener los datos sobre las distancias y las rutas de forma real.

Este apartado refleja los conceptos que hemos aprendido durante la investigación previa al desarrollo del proyecto.

OPTIMIZACIÓN DE RUTAS DE TRANSPORTE

Las rutas de transporte de pasajeros consisten en la recogida de los clientes en puntos prefijados, que llamamos paradas, distribuidas geográficamente a lo largo de un territorio y donde todos estos clientes tienen un punto de destino común prefijado.

La ruta depende de dos parámetros básicamente: el espacio y el tiempo. Con el objetivo de minimizar estos parámetros calcularemos la forma de recorrer las distintas paradas de la ruta.

La eficiencia de estas rutas depende en gran medida de la distribución geográfica de las paradas y su distancia al punto de destino. La finalidad de estas rutas de cara a la empresa es recoger a todos los pasajeros posibles con el menor coste. Por ello cuanta menos distancia recorra el vehículo, menos combustible gastará y más barato será cada viaje. Por ello, el parámetro clave en estas optimizaciones es la distancia recorrida.

En la vida real estas distancias están sujetas a accidentes geográficos y a organizaciones urbanísticas de las diferentes localidades así como a los cambios en la ruta, que hacen que un planteamiento teórico “en línea recta” sea totalmente irreal. Por ello todos los parámetros del sistema deben ser contrastados con un proveedor de mapas online que devuelva datos reales del recorrido. De esta forma se asegura la eficiencia real del recorrido que llevará a cabo el transporte.

El tiempo es otro de los parámetros fundamentales de la optimización de rutas, ya que los pasajeros deben llegar a una hora fijada previamente con el cliente. Este tiempo que tarda en recorrer la distancia de la ruta será inherentemente variable, debido a las numerosas circunstancias que hacen que el tráfico que circula por las vías aumente o disminuya en función del horario de la ruta. Aunque algunos proveedores de mapas online analizan el tráfico y lo calculan en función del momento, esto no es aplicable a nuestro proyecto ya que el recorrido debe ser el mejor a esa hora prefijada, de forma general y aplicable a todos los días laborables. Por lo tanto, la ruta debe tener un margen de tiempo mayor al calculado en la optimización de la propia ruta, para hacer frente a distintas eventualidades a la hora de efectuar el recorrido.

Definiremos ahora las partes que componen una ruta de transporte.

COMPONENTES DE LA GESTIÓN DE RUTAS

Para gestionar adecuadamente una ruta, desde la captura de información de las viviendas de los pasajeros hasta la obtención de la ruta óptima, se deben tener en cuenta los siguientes componentes:

Pasajeros

Son empleados de la empresa que contrata el servicio de transporte. Envían los parámetros de localización (dirección de la vivienda) y turno de trabajo, que son esenciales para la agrupación en paradas. Los pasajeros recibirán un punto y una hora de recogida en función de su posición en la ruta.

Paradas

Las paradas son localizaciones físicas donde se recogen a los pasajeros. Cada parada puede tener varios pasajeros asociados. Cada una de las paradas que se creen deberá estar situada en un punto medio entre las viviendas de todos los pasajeros asociados a esta y en un punto con capacidad para que el vehículo de transporte efectúe la recogida de pasajeros.

Transporte

El transporte puede ser cualquier vehículo con capacidad para albergar a todos los pasajeros que componen la ruta. Deberá recorrer las diferentes paradas en el orden establecido y tratando de mantener el tiempo estipulado tanto de llegada al destino, como de recogida de pasajeros. El conductor del transporte recibirá los detalles de la ruta, en los que se incluye el mapa con las paradas distribuidas y los horarios estipulados.

Vías de la ruta

Indica por donde circulará el transporte. En función del tipo de vía (autovía, vía urbana, etc.) dependerá la velocidad que pueda adquirir el transporte, lo cual afecta directamente al tiempo en el que se pueda completar la ruta.

ESTRUCTURACIÓN DE LOS DATOS PARA SU OPTIMIZACIÓN

Como ya hemos comentado los parámetros principales de nuestro proyecto son la distancia y el tiempo. Estos regirán tanto las funciones de agrupación o Clustering como el algoritmo genético. El Clustering sólo dependerá de la distancia de las paradas entre sí, mientras que el algoritmo genético está determinado por las distancias y por el tiempo que tarda el transporte en recorrerlas. Es importante definir las variables que debe aportar cada componente del gestor de rutas, así como las que debe recibir para poder participar en la optimización. Estas son:

Pasajeros

Parámetros de entrada:

- Dirección de la vivienda.
- Turno de trabajo

Variables de salida:

- Parada asignada
- Hora de recogida

Restricciones:

- El número de pasajeros no debe exceder las plazas del transporte
- Todas las direcciones deben de ser de la misma localidad

Paradas

Parámetros de entrada:

- Dirección de la vivienda.
- El turno de trabajo de cada pasajero.

Variables de salida:

- Localización de cada una de las paradas.
- Hora de recogida de los pasajeros por la ruta.

Restricciones:

- Las paradas deben estar en lugares aptos para efectuar la recogida.
- La distancia máxima para agrupar un pasajero en una parada no será mayor a la estipulada por el administrador.

Transporte

Parámetros de entrada:

- Localización de cada una de las paradas.
- Tiempo de llegada.
- Capacidad máxima del transporte

Variable de salida:

-
- Distancia a recorrer
- Tiempo total en realizar la ruta

Cuando se hayan definido los parámetros de entradas de los pasajeros, el sistema irá calculando progresivamente las variables de salida utilizando los algoritmos de Clustering y genético.

ALGORITMOS DE AGRUPAMIENTO O CLUSTERING

Los métodos de agrupamiento o Clustering constituyen un tipo de aprendizaje por descubrimiento muy similar a la inducción. En el aprendizaje inductivo, un programa aprende a clasificar objetos basándose en etiquetados proporcionados por un profesor (aprendizaje supervisado). En los métodos de agrupamiento no se suministran los datos etiquetados: el programa debe descubrir por sí mismo las clases naturales existentes.

Las funciones de densidad de probabilidad suelen tener una moda o un máximo en una región; es decir, las observaciones tienden a agruparse en torno a una región del espacio de patrones cercana a la moda. Las técnicas de agrupamiento analizan el conjunto de observaciones disponibles para determinar la tendencia de los patrones a agruparse.

Estas técnicas permiten realizar una clasificación asignando cada observación a un agrupamiento (clúster), de forma que cada agrupamiento sea más o menos homogéneo y diferenciable de los demás.

Los agrupamientos naturales obtenidos mediante una técnica de agrupamiento mediante similitud resultan muy útiles a la hora de construir clasificadores cuando no están bien definidas las clases (no existe un conocimiento suficiente de las clases en que se pueden distribuir las observaciones), cuando se desea analiza un gran conjunto de datos (“divide y vencerás”) o, simplemente, cuando existiendo un conocimiento completo de las clases se desea comprobar la validez del entrenamiento realizado y del conjunto de variables escogido.

Los métodos de agrupamiento asocian un patrón a un agrupamiento siguiendo algún criterio de similitud entre individuos. Tales medidas de similaridad deben ser aplicables entre pares de individuos, entre un individuo y un agrupamiento y, finalmente, entre pares de agrupamientos. Generalmente, como medidas de agrupación se emplean métricas de distancia. Algunas de las más habituales son la distancia euclídea, simple, normalizada o ponderada o la distancia de Mahalanobis.

TIPOS DE ALGORITMOS DE CLUSTERING

Los agrupamientos detectados dependen del algoritmo empleado, del valor dado a sus parámetros, de los datos utilizados y de la medida de similitud adoptada.

Se han propuesto cientos de algoritmos de agrupamiento más o menos específicos.

Según se use o no una función criterio se distinguen los algoritmos directos o constructivos (basados en aproximaciones heurísticas) de los algoritmos indirectos o por optimización.

La estructura general de un algoritmo iterativo de agrupamiento directo es la siguiente:

```
Seleccionar una partición inicial
del conjunto de ejemplos en K clusters.

Calcular los centros de los clusters

Mientras que no se estabilicen los clusters

      Repetir

            Generar una nueva partición
            (asignando cada patrón al cluster más cercano)

            Calcular los centroides de los nuevos clusters

      Hasta que se obtenga un valor óptimo de la función de evaluación

      Ajustar el número de clusters
      mezclando y separando clusters existentes
      o eliminando clusters pequeños o "periféricos"
```

Ilustración 4 - Algoritmo de agrupamiento iterativo

La exposición de los algoritmos de Clustering concluirá con un método basado en grafos que utiliza la matriz de similaridad para construir los distintos agrupamientos. Éste método es muy fácil de entender aunque, como sucede con todos los métodos basados en grafos, consume demasiados recursos.

Algunas medidas de distancia utilizadas frecuentemente

1. *Distancia euclídea*: $d^2(X_i, X_j) = (X_i - X_j)^T (X_i - X_j) = \left\| X_i - X_j \right\|^2$

2. *Distancia euclídea normalizada*: Igual que la anterior, salvo que los valores de cada variable se normalizan en el intervalo [0,1] para que unas características no influyan más que otras al calcular las distancias.

3. *Distancia euclídea ponderada*: $d_w^2(X_i, X_j) = \sum_{k=1}^{d} \frac{1}{\sigma_k^2}(X_{ki} - X_{kj})^2$

4. *Distancia de Mahalanobis*: $r^2(X_i, X_j) = (X_i - X_j)^T \Sigma^{-1} (X_i - X_j)$
   donde Σ^{-1} es la inversa de la matriz de covarianza.

Ilustración 5 - Cálculos de distancias entre dos puntos

El algoritmo que hemos diseñado para esta aplicación está basado en varios de los diferentes tipos de algoritmos de agrupamiento que existen.

Método adaptativo

El método adaptativo es un algoritmo heurístico de agrupamiento que se puede utilizar cuando no se conoce de antemano el número de clases del problema. Entre sus ventajas se encuentran su simplicidad y eficiencia. Además, las observaciones se procesan secuencialmente.

Por desgracia, su comportamiento está sesgado por el orden de presentación de los patrones y presupone agrupamientos compactos separados claramente de los demás.

El primer agrupamiento se escoge arbitrariamente. Se asigna un patrón a un clúster si la distancia del patrón al centro del clúster no supera un umbral. En caso contrario, se crea un nuevo agrupamiento.

Este algoritmo incluye una clase de rechazo a la hora de clasificar observaciones. Los patrones se asignan por la regla de mínima distancia. Algunos patrones no son clasificados si el cuadrado de la distancia al agrupamiento más cercano es mayor que el umbral.

Parámetros

τ	Umbral de distancia (al cuadrado)
θ	Fracción (entre 0 y 1)
$\{X_i\}$	Conjunto de patrones

Variables

A	Número actual de agrupamientos
Z_i	Centroide del agrupamiento i

Algoritmo de agrupamiento

Inicialización: $A=1$, $Z_1=X_1$

Mientras queden patrones por asignar

Obtener el siguiente patrón X y calcular $d(X,Z_i)$ $i=1..A$.

Asignar X al agrupamiento más cercano Z_i si $d(X,Z_i) \leq \theta\tau$

Formar un nuevo agrupamiento con X si $d(X,Z_i)>\tau$: $A++$, $Z_A=X$

Recalcular el centroide Z_i y la varianza C_i del agrupamiento.

Ilustración 6 - Algoritmo de agrupamiento adaptativo

Algoritmo de los k medias

El algoritmo de las K medias(o K-means) es probablemente el algoritmo de agrupamiento más conocido. Es un método de agrupamiento heurístico con número de clases conocido (K). El algoritmo está basado en la minimización de la distancia interna (la suma de las distancias de los patrones asignados a un agrupamiento al centro de dicho agrupamiento). De hecho, este algoritmo minimiza la suma de las distancias al cuadrado de cada patrón al centro de su agrupamiento.

El algoritmo es sencillo y eficiente. Además, procesa los patrones secuencialmente (por lo que requiere un almacenamiento mínimo). Sin embargo, está sesgado por el orden de presentación de los patrones (los primeros patrones determinan la configuración inicial de los agrupamientos) y su comportamiento depende enormemente del parámetro K.

Seleccionar arbitrariamente una configuración inicial de los clusters

Repetir

Calcular los centros de los clusters Z_j

Redistribuir los patrones entre los clusters utilizando la mínima distancia euclídea al cuadrado como clasificador:
$X_i \in C_j \leftrightarrow || X_i - Z_j ||^2 < || X_i - Z_l ||^2 \ \forall\, l \neq j$

Hasta que no cambien los centros de los clusters

Ilustración 7 - Algoritmo de las K medias

Métodos basados en grafos: Matriz de similaridad

El principal inconveniente de la mayor parte de los algoritmos heurísticos es su dependencia respecto al orden en que se le presentan los patrones. Los métodos basados en grafos, igual que los algoritmos GRASP, intentan evitar este hecho pero su coste computacional los hace inaplicables en muchas ocasiones.

La matriz de similaridad se emplea para mostrar el grado de similaridad entre un conjunto de patrones. Se construye una matriz S simétrica de tamaño NxN, siendo N el número de patrones del conjunto de entrenamiento. S[i,j] toma el valor 1 si la distancia entre los patrones i y j queda por debajo de un umbral preestablecido. En caso contrario, S[i,j] vale 0. Por lo tanto, con un bit por celda podemos almacenar la matriz de similaridad.

```
Mientras queden patrones en la matriz de similaridad S

      Seleccionar la fila i de la matriz de similaridad S que contenga más
      unos. Si hay varias, escoger una al azar.

      Crear un agrupamiento con los patrones j tales que S[i,j] = 1

      Añadir al agrupamiento todos aquellos patrones k tales que S[j,k] = 1,
      siendo j un patrón incluido en el nuevo agrupamiento hasta que no se
      puedan añadir más patrones a dicho agrupamiento.

      Reducir la matriz de similaridad: Eliminar de S todas las filas y
      columnas correspondientes a patrones incluidos en el agrupamiento
      recién creado.
```

Ilustración 8 - Algoritmo de agrupamiento por matriz de similaridad

PROGRAMACIÓN EVOLUTIVA

Muchos problemas de optimización que aparecen en los ámbitos de las ingenierías son muy difíciles de solucionar por medio de técnicas tradicionales, por lo que a menudo se aplican algoritmos evolutivos, inspirados en la naturaleza, que recogen un conjunto de modelos basados en la evolución de los seres vivos.

El inicio de la utilización de las estrategias evolutivas en la solución de este tipo de problemas data del año 1960 cuando John Holland planteó la posibilidad de incorporar los mecanismos naturales de selección y supervivencia a la resolución de problemas de Inteligencia Artificial ("Adaptation in Natural and Artificial Systems"). Esta investigación fue fundamentalmente académica, siendo su realización práctica en aquella época muy difícil. La simulación de procesos de evolución natural de las especies da como resultado una técnica de optimización estocástica que posteriormente fue llamada algoritmos evolutivos, y que fueron enmarcados dentro de las técnicas no convencionales de optimización para problemas del mundo real. A partir de la creación de estas estrategias evolutivas aparecieron otras vías de investigación como son:

Algoritmos Genéticos (Goldberg), Programación Genética (Koza), Programación Evolutiva (Fogel), y Estrategias de Evolución (Rechenberg/Schwefel), en donde la estrategia de evolución más conocida hoy en día son los algoritmos genéticos.

Los algoritmos genéticos constituyen una técnica poderosa de búsqueda y optimización con un comportamiento altamente paralelo, inspirado en el principio darviniano de selección natural y reproducción genética. En este principio de selección de los individuos más aptos, tienen mayor longevidad y por tanto mayor probabilidad de reproducción. Los individuos descendientes de estos individuos tienen una mayor posibilidad de transmitir sus códigos genéticos a las próximas generaciones.

Las aplicaciones de los algoritmos genéticos han sido muy conocidas en áreas como diseño de circuitos, cálculo de estrategias de mercado, reconocimiento de patrones, acústica, ingeniería aeroespacial, astronomía y astrofísica, química, juegos, programación y secuenciación de operaciones, contabilidad lineal de procesos, programación de rutas, interpolación de superficies, tecnología de grupos, facilidad en diseño y localización, transporte de materiales y muchos otros problemas que involucran de alguna manera procesos de optimización.

ALGORITMOS GENÉTICOS

Los algoritmos genéticos son estrategias de búsqueda estocástica basados en el mecanismo de selección natural y en algunos casos se involucran aspectos de genética natural, imitando a la evolución biológica como estrategia para resolver problemas Los algoritmos genéticos difieren de las estrategias de búsqueda convencionales en que estos (los AGs) trabajan sobre un conjunto de potenciales soluciones, llamado población. Esta población está compuesta de una serie de soluciones llamadas individuos y un individuo está conformado por una serie de posiciones que representan cada una de las variables involucradas en los procesos de optimización y que son llamados cromosomas. Estos cromosomas están compuestos por una cadena de símbolos que en muchos casos esta presentada en números binarios. En un algoritmo genético cada individuo está definido como una estructura de datos que representa una posible solución del espacio de búsqueda del problema.

Las estrategias de evolución trabajan sobre los individuos, que representan las soluciones del problema, por lo que estos evolucionan a través de generaciones. Dentro de la población cada individuo es diferenciado de acuerdo con su valor de aptitud, que es obtenido usando algunas medidas de acuerdo con el problema a resolver. Para la obtención de las próximas generaciones se crean nuevos individuos, llamados hijos, utilizando dos estrategias de evolución básicas como son el operador de cruce y el de mutación (empleadas generalmente de forma aleatoria).

Una nueva generación es obtenida mediante la utilización del operador de selección, que básicamente se realiza sobre los valores de aptitud de los individuos, sobre la población obtenida tras el cruce y la mutación. El operador de selección debe mantener constante el número de individuos de la población, y a través de las generaciones el individuo con mayor valor de aptitud tiene mayores posibilidades de ser seleccionado en la siguiente generación. Entre el conjunto de soluciones candidatas generadas aleatoriamente muchas no funcionarán en absoluto y serán eliminadas; sin embargo, por puro azar, unas pocas pueden resultar prometedoras y pueden mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema. Estas candidatas prometedoras se conservan y se les permite reproducirse. Se realizan múltiples copias de ellas, pero las copias no son perfectas; se introducen cambios aleatorios durante el proceso de copia. Esta descendencia prosigue con la siguiente generación, formando un nuevo acervo de soluciones candidatas que son nuevamente sometidas a una ronda de evaluación de aptitud. Las

candidatas que han empeorado o no han mejorado con los cambios en su código son eliminadas de nuevo; pero, de nuevo por puro azar, las variaciones aleatorias introducidas en la población pueden haber mejorado a algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes.

De nuevo, se seleccionan y copian estos individuos vencedores hacia la siguiente generación con cambios aleatorios, y el proceso se repite. Las expectativas son que la aptitud media de la población se incrementará en cada ronda y, por tanto, repitiendo este proceso cientos o miles de rondas, pueden descubrirse soluciones muy buenas del problema; en otras palabras, tras varias iteraciones el algoritmo converge al individuo con mejor valor de aptitud, el cual representara el óptimo o subóptimo del problema.

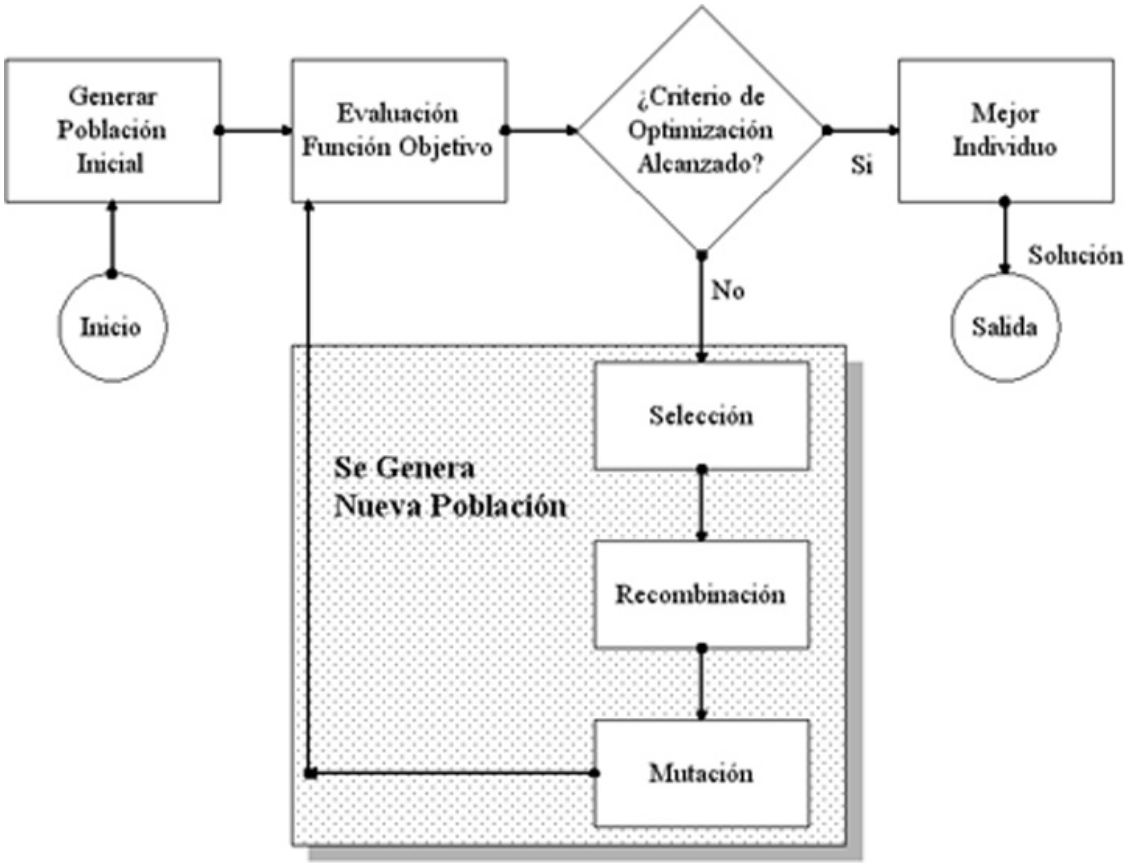

Ilustración 9 - Diagrama de flujo de algoritmo genético

Vocabulario en Algoritmos Genéticos

Los algoritmos genéticos toman para su definición terminología utilizada en Genética Natural y Ciencia Computacional, por lo que la literatura específica es una combinación entre los términos naturales y los artificiales, por esto se debe tener en cuenta:

- La Estructura de codificación con la cual se construyen las soluciones para el problema se llaman cromosomas. Uno o más cromosomas pueden ser requeridos para conformar una población.
- El conjunto completo de cromosomas se llama genotipo y un individuo en particular, un organismo resultante, se llama fenotipo.
- Cada cromosoma contiene una serie de valores individuales llamados genes.
- Cada gen contiene información valiosa de una variable en particular y su ubicación dentro del individuo se conoce como loco. Los diferentes valores de un gen son llamados alelos.

Componentes de un Algoritmo Genético

Los algoritmos genéticos se pueden caracterizar a través de los siguientes componentes:

Problema a optimizar:

Los algoritmos genéticos tienen su campo de aplicación importante en problemas de optimización complejos, donde se tiene diferentes parámetros o conjuntos de variables que deben ser combinadas para su solución. También se enmarcan en este campo problemas con muchas restricciones y problemas con espacios de búsqueda muy grandes. Este tipo de optimización difiere en muchos aspectos de los procesos convencionales de optimización, ya que los algoritmos genéticos:

- Trabajan sobre un conjunto codificado de soluciones, no sobre las soluciones mismas.
- Trabajan sobre un conjunto de soluciones, no sobre una solución única.
- Utilizan información simple para determinar la aptitud de los individuos, dejando de lado los procesos de derivación u otra información adicional.
- Usan procesos de transición probabilística para la búsqueda de la solución, no reglas determinísticas.

Gracias a estas características, los algoritmos genéticos no requieren información matemática adicional sobre optimización, por lo que pueden tomar otro tipo de funciones objetivo y todo tipo de restricciones (lineales y no lineales) definidas sobre espacios discretos, continuos o espacios de búsqueda combinados.

La robustez de los operadores de evolución hace muy efectivos a los algoritmos genéticos en procesos de búsqueda de soluciones globales. A diferencia de las soluciones obtenidas con un algoritmo genético, las búsquedas tradicionales operan sobre una solución particular del problema, buscando soluciones óptimas en las cercanías de esta solución particular, en muchos casos no encontrando soluciones globales para el problema. Por ello, los algoritmos genéticos proveen una gran flexibilidad para ser combinados con heurísticas específicas para la solución de un problema en particular.

Representación para la solución del problema:

Para la solución de un problema en particular es importante definir una estructura del cromosoma de acuerdo con el espacio de búsqueda. En el caso de los algoritmos genéticos la representación más utilizada es la representación binaria, debido a su facilidad de manipulación por los operadores genéticos, su fácil transformación en números enteros o reales, además de la facilidad que da para la demostración de teoremas.

La codificación de números reales en cadenas de números binarios se debe hacer teniendo en cuenta el siguiente proceso y la precisión requerida para la solución del problema.

Decodificación del cromosoma:

Para determinar el número real que la cadena binaria de caracteres representa. Es importante tener en cuenta que la cadena no representa un número real, sino que este número binario etiqueta un número dentro del intervalo inicialmente fijado.

Evaluación de un individuo:

Nos muestra el valor de aptitud de cada uno de los individuos. Esta aptitud viene dada por una función que es la unión entre el mundo natural y el problema a resolver matemáticamente. Esta función de aptitud es particular para cada problema particular a resolver y representa para un algoritmo genético lo que el medio ambiente representa para los humanos.

Operadores Evolutivos: Selección

Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación, pero abajo se listan algunos de los más comunes. Algunos de estos métodos son mutuamente exclusivos, pero otros pueden utilizarse en combinación, algo que se hace a menudo.

Las diferentes variantes en la operación de selección son:

Selección elitista: se garantiza la selección de los miembros más aptos de cada generación. (La mayoría de los AGs no utilizan elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor).

Selección proporcional a la aptitud: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

Selección por rueda de ruleta: una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. (Conceptualmente, esto puede representarse como un juego de ruleta: cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada vez se elige al individuo que “posea” la sección en la que se pare la ruleta).

Selección escalada: al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.

Selección por torneo: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.

Selección por rango: a cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que

reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

Selección generacional: la descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.

Selección por estado estacionario: la descendencia de los individuos seleccionados en cada generación vuelve al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

Selección jerárquica: los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

Operadores Genéticos: Cruce y Mutación

Una vez que la selección ha elegido a los individuos aptos, éstos deben ser alterados aleatoriamente con la esperanza de mejorar su aptitud para la siguiente generación.

Existen dos estrategias básicas para llevar esto a cabo. La primera y más sencilla se llama mutación. Al igual que una mutación en los seres vivos cambia un gen por otro, una mutación en un algoritmo genético también causa pequeñas alteraciones en puntos concretos del código de un individuo.

El segundo método se llama cruce, e implica elegir a dos individuos para que intercambien segmentos de su código, produciendo una "descendencia" artificial cuyos individuos son combinaciones de sus padres. Este proceso pretende simular el proceso análogo de la recombinación que se da en los cromosomas durante la reproducción sexual. Las formas comunes de cruzamiento incluyen al cruzamiento de un punto, en el que se establece un punto de intercambio en un lugar aleatorio del genoma de los dos individuos, y uno de los individuos contribuye todo su código anterior a ese punto y el otro individuo contribuye todo su código a partir de ese punto para producir una descendencia, y al cruzamiento uniforme, en el que el valor de una posición dada en el genoma de la descendencia corresponde al valor en esa posición

del genoma de uno de los padres o al valor en esa posición del genoma del otro padre, elegido con un 50% de probabilidad.

Los operadores genéticos en este caso son operadores con memoria, guardando en cada iteración los códigos genéticos de los mejores individuos a través de las generaciones. Los procesos de evolución darviniana generalmente son procesos miméticos a través de las generaciones.

Ventajas de los Algoritmos genéticos

El primer y más importante punto es que los algoritmos genéticos son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta subóptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los AGs tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución. Otra ventaja del paralelismo es que, al evaluar la aptitud de una cadena particular, un algoritmo genético sondea al mismo tiempo cada uno de los espacios a los que pertenece dicha cadena. Tras muchas evaluaciones, iría obteniendo un valor cada vez más preciso de la aptitud media de cada uno de estos espacios, cada uno de los cuales contiene muchos miembros. Por tanto, un AG que evalúe explícitamente un número pequeño de individuos está evaluando implícitamente un grupo de individuos mucho más grande. De la misma manera, el AG puede dirigirse hacia el espacio con los individuos más aptos y encontrar el mejor de ese grupo. En el contexto de los algoritmos evolutivos, esto se conoce como teorema del esquema, y es la ventaja principal de los AGs sobre otros métodos de resolución de problemas.

Debido al paralelismo que les permite evaluar implícitamente muchos esquemas a la vez, los algoritmos genéticos funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande, demasiado vasto para hacer una búsqueda exhaustiva en un tiempo razonable. La mayoría de los problemas que caen en esta categoría se conocen como “no lineales”. En un problema lineal, la aptitud de cada componente es independiente, por lo que cualquier mejora en alguna parte dará como resultado una mejora en el sistema completo. No es necesario decir que hay pocos problemas como éste en la vida real. La no linealidad es la norma, donde cambiar un componente puede tener efectos en cadena en todo el sistema, y donde cambios múltiples que, individualmente, son perjudiciales, en combinación

pueden conducir hacia grandes mejoras en la aptitud. La no linealidad reduce una explosión combinatoria en el número de posibilidades. El paralelismo implícito de los AGs les permite superar incluso este enorme número de posibilidades, y encontrar con éxito resultados óptimos o muy buenos en un corto periodo de tiempo, tras muestrear directamente sólo regiones pequeñas del vasto paisaje adaptativo.

Otra ventaja notable de los algoritmos genéticos es que se desenvuelven bien en problemas con un paisaje adaptativo complejo: aquéllos en los que la función de aptitud es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales. La mayoría de los problemas prácticos tienen un espacio de soluciones enorme, imposible de explorar exhaustivamente; el reto se convierte entonces en cómo evitar los óptimos locales, es decir, las soluciones que son mejores que todas las que son similares a ella, pero que no son mejores que otras soluciones distintas situadas en algún otro lugar del espacio de soluciones. Muchos algoritmos de búsqueda pueden quedar atrapados en los óptimos locales. Los algoritmos evolutivos, por otro lado, han demostrado su efectividad al escapar de los óptimos locales y descubrir el óptimo global incluso en paisajes adaptativos muy escabrosos y complejos. Los cuatro componentes principales de los AGs -paralelismo, selección, mutación y cruzamiento- trabajan juntos para conseguir esto.

Otra área en el que destacan los algoritmos genéticos es su habilidad para manipular muchos parámetros simultáneamente. Muchos problemas de la vida real no pueden definirse en términos de un único valor que hay que minimizar o maximizar, sino que deben expresarse en términos de múltiples objetivos, a menudo involucrando contrapartidas: uno sólo puede mejorar a expensas de otro. Los AGs son muy buenos resolviendo estos problemas: en particular, su uso del paralelismo les permite producir múltiples soluciones, igualmente buenas, al mismo problema, donde posiblemente una solución candidata optimiza un parámetro y otra candidata optimiza uno distinto, y luego un supervisor humano puede seleccionar una de esas candidatas para su utilización. Si una solución particular a un problema con múltiples objetivos optimiza un parámetro hasta el punto en el que ese parámetro no puede mejorarse más sin causar una correspondiente pérdida de calidad en algún otro parámetro, esa solución se llama óptimo paretiano o no dominada.

Finalmente, una de las cualidades de los algoritmos genéticos que, a primera vista, puede parecer un desastre, resulta ser una de sus ventajas: a saber, los AGs no saben nada de los problemas que deben resolver. En lugar de utilizar información específica conocida a priori para guiar cada paso y realizar cambios guiados expresamente hacia la mejora, los AGs realizan cambios aleatorios en sus soluciones candidatas y luego utilizan la función de aptitud para determinar si esos cambios producen una mejora.

Como sus decisiones están basadas en la aleatoriedad, todos los caminos de búsqueda posibles están abiertos teóricamente a un AG; en contraste, cualquier estrategia de resolución de problemas que dependa de un conocimiento previo, debe inevitablemente comenzar descartando muchos caminos a priori, perdiendo así cualquier solución novedosa que pueda existir. Los AGs, al carecer de ideas preconcebidas basadas en creencias establecidas sobre "cómo deben hacerse las cosas" o sobre lo que "de ninguna manera podría funcionar", los AGs no tienen este problema. De manera similar, cualquier técnica que dependa de conocimiento previo fracasará cuando no esté disponible tal conocimiento, pero, de nuevo, los AGs no se ven afectados negativamente por la ignorancia. Mediante sus componentes de paralelismo, cruzamiento y mutación, pueden viajar extensamente por el paisaje adaptativo, explorando regiones que algoritmos producidos con inteligencia podrían no haber tenido en cuenta, y revelando potencialmente soluciones asombrosas y creativas.

Limitaciones de los Algoritmos Genéticos

Aunque los algoritmos genéticos han demostrado su eficiencia y potencia como estrategia de resolución de problemas, no son la panacea. Los AGs tienen ciertas limitaciones; sin embargo, se demostrará que todas ellas pueden superarse y que ninguna de ellas afecta a la validez de la evolución biológica.

La primera y más importante consideración al crear un algoritmo genético es definir una representación del problema. El lenguaje utilizado para especificar soluciones candidatas debe ser robusto; es decir, debe ser capaz de tolerar cambios aleatorios que no produzcan constantemente errores fatales o resultados sin sentido. Hay dos maneras principales para conseguir esto. La primera, utilizada por la mayoría de los algoritmos genéticos, es definir a los individuos como listas de números -binarios, enteros o reales- donde cada número representa algún aspecto de la solución candidata. En otro método, la programación genética, el propio código del programa sí cambia. Ambos métodos producen representaciones robustas ante la mutación, y pueden representar muchos tipos diferentes de problema con éxito.

El problema de cómo escribir la función de aptitud debe considerarse cuidadosamente para que se pueda alcanzar una mayor aptitud y verdaderamente signifique una solución mejor para el problema dado. Si se elige mal una función de aptitud o se define de manera inexacta, puede que el algoritmo genético sea incapaz de encontrar una solución al problema, o puede acabar resolviendo el problema equivocado (esta última situación se describe a veces como la tendencia del AG a "engañar", aunque en

realidad lo que está pasando es que el AG está haciendo lo que se le pidió hacer, no lo que sus creadores pretendían que hiciera).

Además de elegir bien la función de aptitud, también deben elegirse cuidadosamente los otros parámetros de un AG, como son el tamaño de la población, el ritmo de mutación y cruzamiento, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el algoritmo genético no explore suficientemente el espacio de soluciones para encontrar buenas soluciones consistentemente. Si el ritmo de cambio genético es demasiado alto o el sistema de selección se escoge inadecuadamente, puede alterarse el desarrollo de esquemas beneficiosos y la población puede entrar en catástrofe de errores, al cambiar demasiado rápido para que la selección llegue a producir convergencia. La solución ha sido “la evolución de la evolutividad”: las adaptaciones que alteran la habilidad de una especie para adaptarse.

Un problema con el que los algoritmos genéticos tienen dificultades son los problemas con las funciones de aptitud “engañosas”, en las que la situación de los puntos mejorados ofrece información engañosa sobre dónde se encuentra probablemente el óptimo global. La solución a este problema es la misma para los algoritmos genéticos y la evolución biológica: la evolución no es un proceso que deba encontrar siempre el óptimo global. Puede funcionar casi igual de bien alcanzando la una solución muy buena aunque no sea la realmente óptima y, para la mayoría de las situaciones, eso será suficiente, incluso aunque el óptimo global no pueda alcanzarse fácilmente desde ese punto.

Un problema muy conocido que puede surgir con un AG se conoce como convergencia prematura. Si un individuo que es más apto que la mayoría de sus competidores emerge muy pronto en el curso de la ejecución, se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto, provocando que el algoritmo converja hacia el óptimo local que representa ese individuo, en lugar de rastrear el paisaje adaptativo lo bastante a fondo para encontrar el óptimo global. Esto es un problema especialmente común en las poblaciones pequeñas, donde incluso una variación aleatoria en el ritmo de reproducción puede provocar que un genotipo se haga dominante sobre los otros. Los métodos más comunes implementados por los investigadores en AGs para solucionar este problema implican controlar la fuerza selectiva, para no proporcionar tanta ventaja a los individuos excesivamente aptos. La selección escalada, por rango y por torneo son tres de los métodos principales para conseguir esto.

Finalmente, varios investigadores aconsejan no utilizar algoritmos genéticos en problemas resolubles de manera analítica. No es que los algoritmos genéticos no

puedan encontrar soluciones buenas para estos problemas; simplemente es que los métodos analíticos tradicionales consumen mucho menos tiempo y potencia computacional que los AGs y, a diferencia de los AGs, a menudo está demostrado matemáticamente que ofrecen la única solución exacta.

WEB MAPPING

Mapeo Web es el proceso de diseño, implementación, generación y muestra de mapas en la Web. Mientras el "mapeo" web se ocupa principalmente de cuestiones tecnológicas, la cartografía web estudia, además, aspectos teóricos: el uso de mapas web, la evaluación y la optimización de las técnicas y los flujos de trabajo, la facilidad de uso de mapas web, aspectos sociales, etc. Todas estas técnicas han surgido como respuesta a la necesidad de las personas y los negocios a tener un mapa interactivo que revele información acerca de la localización de personas y objetos. Un caso especial de mapas web son los mapas móviles, que se muestran en los dispositivos móviles, como los Smartphones, PDAs y GPS.

El uso de la web como un medio de difusión de los mapas puede ser considerado como un gran avance en la cartografía y abre muchas nuevas oportunidades, tales como mapas en tiempo real y contenido personalizado en los mapas, así como el intercambio de información geográfica. También implica muchos desafíos debido a las restricciones técnicas (resolución de pantalla baja y ancho de banda limitado, en particular, con los dispositivos móviles, muchos de los cuales son físicamente pequeños, y el uso de conexiones inalámbricas a Internet lentas), derechos de autor y las cuestiones de seguridad, problemas de fiabilidad y complejidad técnica. Aunque los primeros mapas web eran principalmente estáticos, los mapas web de hoy en día pueden ser completamente interactivos y se pueden integrar en múltiples medios de comunicación. Esto significa que, tanto el mapeo web, como la cartografía web, tienen que lidiar con la interactividad y usabilidad.

En nuestra aplicación hemos usado uno de los "Servicios de Localización" más conocidos. Google Maps. Hemos utilizado este servicio ya que es el más completo en cuanto a opciones de representación y ofrece una amplia variedad de APIs para interactuar con el servicio Web, siendo todo esto gratuito.

GOOGLE MAPS

Google Maps es una aplicación de servicio de Web Mapping cuya tecnología está proporcionada por Google. Alimenta muchos servicios basados en mapas, como el sitio web de Google Maps, Google Ride Finder, Google Transit y los mapas incrustados en páginas web de terceros a través de Google Maps API. Ofrece mapas de calles, un planificador de rutas para viajar a pie (beta), en coche, en bicicleta (beta), o con el transporte público. También es un localizador geográfico de lugares, edificios o comercios en diversos lugares del mundo.

Ilustración 10 - Google Maps

Las imágenes de satélite de Google Maps no se actualizan en tiempo real, sino que se esto se produce con una periodicidad semestral.

El sistema de localización de Google Maps está basado en coordenadas cartográficas. Están en el sistema WGS84 1 y se mostrará la latitud y la longitud, positiva para Norte y Este, negativa para Sur y Oeste.

Algunas de las funcionalidades de Google Maps son:

- Visualización de mapas en formato mapa o captura por satélite
- Búsqueda de direcciones.
- Diseño de rutas entre dos puntos, para automóviles, a pie y en transporte público. Calculo de distancias y de tiempo entre eso dos puntos. Además tiene en cuanta el tráfico en tiempo real y otros modificadores de la circulación normal
- Visualización de imágenes de satélites de toda la geografía.

CAPÍTULO III: LA APLICACIÓN

INTRODUCCIÓN AL CAPÍTULO

En este apartado describiremos cómo funciona y en qué consiste la aplicación de forma detallada. Empezaremos describiendo la base de datos que hemos diseñado y empleado para almacenar los usuarios y las paradas a tratar y por qué nos decantamos por esta opción.

Posteriormente, hablaremos de cómo hemos aplicado los algoritmos de agrupamiento (Clustering) para obtener el menor número de paradas y comprobaremos cómo esto reduce el tiempo y la distancia utilizados por el transporte durante su ruta de recogida.

A continuación, se explicará cómo hemos codificado los datos de la ruta para aplicarles el algoritmo evolutivo. Dentro de este punto, comentaremos también las diferentes funciones de selección, cruce y mutación que hemos empleado y por qué nos hemos decidido por ellas.

Por último, explicaremos la forma de relacionarse con el proveedor de mapas online, recoger la información que envía y la forma de conectar todos los componentes de la aplicación.

BASE DE DATOS

La base de datos de la aplicación la hemos realizado mediante ficheros Excel debido a su facilidad de uso gracias a ciertas bibliotecas de C# que indicaremos más adelante.

Existen dos tipos de ficheros Excel, uno de ellos será utilizado por los pasajeros para pasar la información al administrador de la aplicación, mientras que el otro tipo de fichero se utilizará para almacenar toda la información necesaria para el correcto funcionamiento de la aplicación, obtenida a partir de la suma de la información de los ficheros enviados por los pasajeros y las rutas calculadas mediante los algoritmos implementados.

PLANTILLA DE PASAJERO

Mediante esta plantilla, que se muestra en la Ilustración 11 - Ejemplo Plantilla Excel de los usuarios, los pasajeros registran su información en la aplicación, especialmente la de su lugar de residencia y su turno de trabajo, a partir de la cual se generará la ruta del transporte. El usuario tendrá que rellenar todos los campos y posteriormente enviar el fichero al administrador de la aplicación por correo electrónico. A la hora de rellenar los datos, hay algunos datos preestablecidos y en los que el usuario no podrá escoger libremente si no que se le obligará a elegir entre una serie de opciones. Éste es el caso del Tipo de operación (Alta, Baja o Modificación) o el Turno de trabajo (Mañana, Tarde o Noche).

Una vez recibidos los datos, el administrador colocará en la carpeta apropiada, junto a las demás plantillas recibidas, para que el sistema recupere la información de cada usuario y la guarde en el archivo de base de datos principal.

Aquí podemos ver u ejemplo de información de un usuario:

	A	B	C	D	E	F	G	H
1	Tipo operación	DNI/Pasaporte	Nombre y apellidos	Dirección	Codigo Postal	Num. Telefono	Turno de Trabajo	Minusvalido
2	Alta	70258963R	Andres Aguado	Calle de Andres Mellado 12	28015	555555555	Manana	NO
3	Alta	70255444N	Javier Jimenez de Vega	Calle de Andres Mellado 12	28015	666666666	Manana	NO
4	Modificacion	70255434F	Juan Alcácer	Calle Gran Vía 3	28013	343434344	Manana	NO
5	Baja	70255446H	Alfredo Martínez	Calle Carranza 5	28004	343434434	Manana	NO
6	Alta	70255441H	Elsa Baquerizo	Calle Serrano 10	28001	545584845	Manana	NO

Ilustración 11 - Ejemplo Plantilla Excel de los usuarios

Como podemos ver, se pueden enviar varios usuarios dentro de la misma plantilla de pasajero.

Los campos a rellenar son:

- Tipo de operación: Este campo determina como tratar la información que envía cada pasajero. Si se quiere dar de alta, se introducirá un nuevo pasajero eligiendo la opción “Alta”, si el usuario quiere modificar algún parámetro de su información, elegirá la “Modificación” y por último, si quiere darse de baja en el servicio de ruta, elegirá la opción “Baja”
- DNI: DNI del empleado para diferenciarlo de otros empleados con el mismo nombre.

- Nombre y apellidos: Nombre y apellidos del empleado. Esta información se usará para visualizar los viajeros que se subirán al transporte en cada parada.
- Dirección: Esta será la dirección que se tendrá en cuenta para el cálculo de la ruta.
- Código Postal: Ayuda a identificar la población en la que se encuentra la dirección del pasajero.
- Teléfono: Útil por si se debe contactar con algún pasajero de manera urgente.
- Turno de trabajo: Esto nos servirá para agrupar las personas que participaran en la ruta, ya que comparten el mismo horario de entrada al trabajo.

Una vez se importen los datos al fichero principal de la base de datos, los ficheros enviados por el usuario se eliminarán.

PLANTILLA BASE DE DATOS

La plantilla que sirve para almacenar la información útil para la aplicación consta de una pestaña que contiene toda la información que los usuarios han enviado a través de sus plantillas y una hoja por cada ruta calculada donde se almacenarán las paradas de las mismas y el orden en el que deben recorrerse para hacer la ruta óptima. Hay una ruta por cada turno de trabajo (mañana, tarde y noche).

En la siguiente ilustración se muestra la hoja “Pasajeros”, que recoge la suma de la información que mandan los usuarios, añadiendo a ésta, la ruta y la parada asignada a cada usuario registrado.

	A	B	C	D	E	F	G	H	I
1	DNI/Pasaporte	Nombre y apellidos	Dirección	Codigo Postal	Num. Telefono	Turno de Trabajo	Minusvalido	Num. Ruta	Num. Parada
2	70258963R	Andres Aguado	Calle de Andres Mellado 12	28015	555555555	Manana	NO	1	1
3	70255444N	Javier Jimenez de Vega	Calle de Andres Mellado 12	28015	666666666	Manana	NO	1	2
4	70255434F	Juan Alcácer	Calle Gran Vía 3	28013	343434344	Manana	NO	1	3
5	70255446H	Alfredo Martínez	Calle Carranza 5	28004	343434434	Manana	NO	1	4
6	70255441H	Elsa Baquerizo	Calle Serrano 10	28001	545584845	Manana	NO	1	5

Ilustración 12 - Plantilla Base de Datos. Hoja Pasajeros

En la siguiente imagen, podemos apreciar el formato que tiene cada hoja donde se almacenan las rutas de cada turno de trabajo:

Turno	Mañana		
Num. Ruta	1		
Num. Paradas	4		
	Latitud	Longitud	Hora
Parada 1	40,437827	-3,679537	9:00
Parada 2	40,430607	-3,673739	9:10
Parada 3	40,433036	-3,691567	9:15
Parada 4	40,431105	-3,696643	9:30
Parada 5	41,217984	-4,524221	9:35
Parada 6			
Parada 7			
Parada 8			
Parada 9			
Parada 10			
Parada 11			
Parada 12			
Parada 13			
Parada 14			
Parada 15			
Parada 16			
Parada 17			

Ilustración 13 - Plantilla Base de Datos. Hoja Rutas

En cada una de estas pestañas guardamos el número de paradas que se han generado para dar servicio a todos los pasajeros que aparecen en la pestaña de “Pasajeros” antes mencionada. Además, aparecerá el turno de la ruta que puede tomar los valores: Mañana, Tarde o Noche. A cada turno se le asignan rutas independientes.

Cada parada tendrá asignadas unas coordenadas de latitud y longitud, que servirán para enviarlas al proveedor de mapas y recuperar la dirección a la hora de representar los puntos en el mapa. También tendrá asignada un número de parada que actuará como identificador junto con el turno de la ruta.

Cada vez que se carga el programa, se examina la carpeta pensada para que el administrador deposite los archivos enviados por email en busca de nuevos archivos que contengan datos de nuevos pasajeros. En el caso de que haya nueva información, se recalcularán todas las agrupaciones de paradas y también el orden de recorrerlas

mediante el algoritmo genético, debido a que un pasajero más o menos podría hacer variar la forma de optimizar la ruta. La base de datos siempre tendrá que estar encuadrada en la propia carpeta de desarrollo del proyecto para su correcto funcionamiento, junto con las platillas que envían los usuarios.

VENTAJAS DE MICROSOFT EXCEL COMO BASE DE DATOS

Utilizando este formato de base de datos, no sólo se facilita el uso de la misma a los usuarios que no posean conocimientos técnicos, ya que el Excel es una herramienta intuitiva y de uso muy extendido, además, permite al administrador disponer de la información de toda la aplicación, de forma clara y sencilla sin ser necesario que éste posea conocimientos de Bases de Datos o que se haya creado un procedimiento para extraer la información. En cualquier momento, y con una herramienta tan familiar como Excel, se puede recuperar cualquier dato concreto de la base de datos. Esto agilizará tareas de mantenimiento y corrección de errores, ya que no se necesitará personal cualificado para la realización de las mismas.

Además, utilizando para el desarrollo de la aplicación Visual Studio como lo hemos hecho, existen librerías que facilitan la operación sobre ficheros Excel, por lo que, la forma de escribir, leer y, en definitiva, manipular estos ficheros se hace bastante sencilla.

ALGORITMO DE CLUSTERING

El algoritmo de agrupamiento de pasajeros en paradas (Clustering), está basado en una mezcla de los diversos tipos de Clustering que mostrábamos en la descripción de éstos anteriormente.

La dirección de las casas está representada en coordenadas geográficas de longitud y latitud que se habrán conseguido a través del Servicio Web de mapas "Google Maps". La distancia entre paradas y casas está calculada en distancia euclídea, esto se realiza durante nuestro algoritmo de Clustering, ya que de esta manera se ahorra numerosos accesos al servicio de mapas, lo que ralentizaría la recogida de datos al estar pidiendo siempre las distancias al propio servicio. Además, como son distancias que el pasajero recorre a pie, no está sujeto a normas de circulación y puede atravesar obstáculos que los vehículos no, como parques, calles peatonales, direcciones prohibidas...

El algoritmo recibe las viviendas de los pasajeros que habrá que agrupar y, en la primera iteración, se creará una parada para cada vivienda. A continuación, el algoritmo crea una matriz de distancias entre las viviendas y las paradas, si todas las casas tienen una parada a menos de la distancia estipulada como límite, se dan como buenas las paradas y se realiza otra agrupación sobre las mismas, volviendo a calcular la matriz de distancias. Si alguna casa no cuenta con una parada de la ruta lo suficientemente cerca, se desecha esa agrupación y se retorna como agrupación final el conjunto de paradas anterior.

```
class Clustering
  {
     //Matrices que almacenaran las coordenadas de las paradas y las casas
     private Double[][] puntos = new Double[2][], casas = new Double[2][]; //0 Latitud, 1 Long

      private int[] casasParadaTemp;
      public Clustering(PointLatLng[] coordenadas) {
        puntos[0] = new Double[coordenadas.Length];
        puntos[1] = new Double[coordenadas.Length];

        casas[0] = new Double[coordenadas.Length];
        casas[1] = new Double[coordenadas.Length];

        casasParadaTemp = new int[coordenadas.Length];

        for (int i = 0; i<coordenadas.Length;i++)
        {
           puntos[0][i] = coordenadas[i].Lat;
```

```
            puntos[1][i] = coordenadas[i].Lng;
            casas[0][i] = coordenadas[i].Lat;
            casas[1][i] = coordenadas[i].Lng;
        }
        puntos = clusterParadas(); //De aquí saldrá en la matriz puntos,
                                   //los puntos de cada parada y en la matriz casas las paradas que les
                                    corresponde
    }
```

Una vivienda deberá tener una parada de la ruta como máximo a 700 metros de ella. Esta distancia se ha calculado para que no resulte excesiva de andar para los pasajeros y que no sea tan pequeña como para entorpecer la función del Clustering.

```
    //Calcula distancia entre 2 puntos de forma euclidiana
    public Double distancia(Double lat1, Double lng1, Double lat2, Double lng2)
    {
        return Math.Sqrt(Math.Pow(lat1 - lat2, 2) + Math.Pow(lng1 - lng2, 2));
    }
```

El algoritmo iterará un número de veces igual al número de casas para ir agrupando, dos a dos, todas las casas que estén cercanas, en paradas. De esta forma, cada vez que se hace una iteración y se hayan agrupado dos casas en una parada, se vuelve a crear una matriz de distancias y la de agrupaciones, comprobando que en esta solución parcial no haya ninguna casa que esté a una distancia superior a 700 metros, de su lugar de recogida. Estas iteraciones seguirán hasta que no se produzca ninguna nueva agrupación, lo que devolverá la solución. De esta forma, podría generarse paradas inservibles, esto es que no sea la más cercana a ninguna vivienda a pesar de que parta de ellas. Por ello, al final del algoritmo se recorrerán todas las paradas para descartar las que no tengan viajeros que acudan a ellas.

La parada siempre se pone en la mediatriz de la recta que cruza los dos puntos agrupados, para que la distancia de las viviendas de los pasajeros a las paradas sea equitativa.

El siguiente fragmento de código muestra la función principal del algoritmo de Clustering; se encarga de, a partir de las viviendas, si es la primera iteración, o de las paradas resultantes de la iteración anterior, calcular las nuevas paradas y calcular si éstas son correctas.

```
    //Calcular nuevas paradas
    private Double[][] clusterParadas()
    {
```

```
//Matriz de distancias entre las casas y las paradas
Double[][] matrizDistancias = new Double[casas[0].Length][];
//Matriz auxiliar para guardar temporalmente las paradas
Double[][] puntosAux = new Double[2][];
bool flag = true; //Para controlar si se hacen cambios.
//Se agruparán paradas tantas veces como casas haya o hasta que no haya cambios.
for (int iteraciones = 0; iteraciones < casas[0].Length && flag; iteraciones++)
{
   puntosAux[0] = new Double[puntos.Length];
   puntosAux[1] = new Double[puntos.Length];
   //Creamos la matriz de distancias en funcion del número de paradas que haya
   for (int i = 0; i < casas[0].Length; i++)
   {
      matrizDistancias[i] = new Double[puntos[0].Length];
   }

   //Para cada casa
   for (int i = 0; i < casas[0].Length; i++)
   {
      //Calculamos la distancia a cada parada
      for (int j = i; j < puntos[0].Length; j++)
      {
         matrizDistancias[i][j] = distancia(casas[0][i], casas[1][i], puntos[0][j], puntos[1][j]);
         matrizDistancias[j][i] = matrizDistancias[i][j];
      }
      //Si hay más de 700m desde la casa a la parada, devolvemos el grupo de paradas
    anteriores.
      if (!paradasCorrectas(matrizDistancias))
      {
         return puntosAux;
      }
   }

   //Si solo queda una parada, comprobamos si es correcta y salimos devolviendo la
  parada si lo es o la anterior agrupación si no.
   if (puntos[0].Length == 1)
   {
      if (paradasCorrectas(matrizDistancias))
      {
         return puntos;
      }
      else
      {
         return puntosAux;
      }
   }

   //Si las paradas actuales son correctas, intentamos agrupar más.
```

```
            //Salvamos las paradas actuales para devolverlas en caso de que la siguiente
           agrupación sea incorrecta.
            puntosAux = (Double[][])puntos.Clone();

            //Variable temporal para almacenar distancias cogemos la distancia entre la 1 y la 2
           como referencia
            Double temp;
            //Array para indicar qué paradas están agrupadas y con cuál
            int[][] agrupaciones = new int[2][];
            agrupaciones[0] = new int[puntos[0].Length];
            agrupaciones[1] = new int[puntos[0].Length];
            int contParadas=0;
            //Ponemos todas las agrupaciones a -1 para que coja la 0 tb.
            for (int i = 0; i < agrupaciones[0].Length; i++)
            {
               agrupaciones[0][i] = -1;
               agrupaciones[1][i] = -1;
            }

            //Para cada parada, miramos la más cercana de las que no estén aún agrupadas
            for (int i = 0; i < puntosAux[0].Length;i++ )
            {
               bool aux = false;
               temp = 500;
               //Si la parada a examinar no está cogida
               if(!agrupaciones[0].Contains(i) && !agrupaciones[1].Contains(i)){
                  //Para cada parada
                  for(int j=i;j<puntos[0].Length;j++){

                        //Si no están agrupadas y la distancia a esa parada es menor que a la
                        primera, se agrupan las paradas
                     if( i!=j && !agrupaciones[0].Contains(j) && !agrupaciones[1].Contains(j) &&
temp >= distancia(puntosAux[0][i], puntosAux[1][i], puntosAux[0][j], puntosAux[1][j])){
                        agrupaciones[0][contParadas] = i; //Emparezamos la parada i
                        agrupaciones[1][contParadas] = j; //Con la j
                        temp = distancia(puntosAux[0][i], puntosAux[1][i], puntosAux[0][j],
puntosAux[1][j]);
                        aux = true;
                     }
                  }
                  //Si no hemos encontrado parada cercana, se empareja la parada consigo misma
                  if (!aux)
                  {
                     agrupaciones[0][contParadas] = i;
                     agrupaciones[1][contParadas] = i;
                  }
                  contParadas++; //Contamos la parada
               }
            }
```

```
            //Una vez agrupadas las paradas, calculamos las nuevas y lo almacenamos en el array
            puntos.
            puntos = new Double[2][];
            puntos[0] = new Double[contParadas];
            puntos[1] = new Double[contParadas];
            for (contParadas = contParadas-1; contParadas >= 0; contParadas--)
            {
                puntos[0][contParadas] =
(puntosAux[0][agrupaciones[0][contParadas]]+puntosAux[0][agrupaciones[1][contParadas]])/2;
                puntos[1][contParadas] = (puntosAux[1][agrupaciones[0][contParadas]] +
puntosAux[1][agrupaciones[1][contParadas]]) / 2;
            }
        }

        //Si llega aquí es que o se han acabado las iteraciones o en esta agrupación no hay
cambios.
        //Devolvemos la última agrupación ya que o será la más reciente y más correcta que la
anterior, o será igual a la anterior
        return puntos;
    }
```

El algoritmo, al terminar, devolverá un array donde estarán las paradas definitivas con las agrupaciones de las viviendas de pasajeros. De esta forma nos aseguramos que el transporte no se tenga que desplazar expresamente a cada vivienda de cada usuario.

El orden en el que se almacenan las paradas en el array aún no es el orden en el que deben recorrerse las paradas; es un orden aleatorio. Este array será la entrada del algoritmo genético que se encargará de calcular el orden en el que éstas deberán recorrerse para hacer el camino óptimo.

ALGORITMO GENÉTICO

El algoritmo genético que hemos implementado sigue la estructura habitual de codificación de datos en genes, cromosomas y poblaciones. Sobre ellos, se aplican los operadores de selección, cruce y mutación. Además incluiremos elitismo, para asegurarnos de que no perdemos el mejor individuo de cada iteración. Seguidamente, describiremos cada uno de estos pasos detalladamente, incluyendo la función de evaluación de individuos y los criterios que hemos seguido.

CODIFICACIÓN DE DATOS EN GENES

Cada gen será una parada, codificada como un punto geográfico de latitud y longitud, que ayudará a que el paso de direcciones y cálculos que tenga que hacer el Servicio Web de mapas sea lo más rápido posible y no tenga ambigüedades.

Generación de genes, cromosomas y creación de poblaciones

La codificación del problema se basa en escoger cuál será la base del algoritmo, es decir, los genes. En este caso, hemos decidido que cada parada de la ruta sea un gen.

Cada cromosoma estará compuesto por un array de genes que incluirá todos los genes (paradas) y el orden de éstos dentro del array indicará el orden en el que se visita cada parada. Esto será un requisito indispensable a lo largo de todo el algoritmo y se deberán realizar comprobaciones en diversos puntos del mismo para comprobar que los cromosomas sigan siendo válidos. Para ayudar a la generación de cromosomas diferentes, ordenamos el array de genes de forma aleatoria, lo cual aumenta la diversidad de la población y produce mejores resultados al final del algoritmo. El mejor individuo obtenido será la solución del algoritmo.

A partir de los cromosomas, se generan las poblaciones. Cada población tendrá un tamaño de 200 cromosomas para que se aplique a ellos el algoritmo genético y, el resultado de cada iteración del algoritmo generará una nueva población.

El resultado del algoritmo será el mejor cromosoma que se haya obtenido en cualquiera de las iteraciones del algoritmo.

FUNCIÓN DE EVALUACIÓN

Esta función calcula y asigna una puntuación a cada cromosoma para, de esta forma, poder comparar individuos y hallar el mejor. Para ello hemos creado una función que calcula las distancias parciales entre todos los puntos en el orden en el que estén en el array y, posteriormente, se suman todas, junto con la distancia entre la última parada y el destino de la ruta. Para que la evaluación tenga un sentido positivo, es decir, a mayor evaluación, mejor individuo, se resta a un número suficiente grande el resultado de esta suma, dando lugar al resultado de la evaluación.

En este caso sí que hacemos uso del servicio web para hallar las distancias, a diferencia del algoritmo de Clustering que lo hacíamos con la distancia euclidiana, ya que refleja un tiempo mucho más realista para un vehículo.

```
public Double getEvaluacion()
      {
         Double evaluacion = 0;
         MapRoute route;
         MainForm aux = new MainForm(1);
         RoutingProvider rp = aux.MainMap.MapProvider as RoutingProvider;
         String[] distanciaString;
         String aux2;

         for (int i = 0; i < this.genes.Length-1;i++ )
         {
            aux2 = "";
            route = rp.GetRoute(this.genes[i].getPoint(), this.genes[i + 1].getPoint(), false, false,
(int)aux.MainMap.Zoom);
            distanciaString = route.Name.Trim().Split(' ');

            for (int m = 1; m < distanciaString[0].Length; m++)
               aux2 += distanciaString[0][m];

            if (distanciaString[1] == "km")
            {
               if (aux2.Contains("."))
                  evaluacion += (Convert.ToDouble(aux2) / 10);
               else
                  evaluacion += (Convert.ToDouble(aux2));
            }
            else
            {
               evaluacion += (Convert.ToDouble(aux2) / 1000);
            }
         }
          evaluacion = 500- evaluacion;
```

```
        aux.Close();
        return evaluacion;
    }
```

FUNCIONES DE SELECCIÓN

Para las funciones de selección, nos hemos decantado por usar diferentes tipos de funciones de selección distribuidas de forma proporcional en función al número de iteraciones; de esta forma nos aseguramos de que nos beneficiamos de las ventajas de todas ellas aplicándolas a las diferentes poblaciones en cada iteración. A continuación, mostraremos los algoritmos que implementan cada función de selección. No las describiremos ya que, tanto su funcionamiento, como la estructura de los algoritmos, lo hicimos anteriormente.

Ruleta

```
public Poblacion ruleta(Poblacion pob){
        Poblacion aux = new Poblacion(pob.getIndividuos().Length);
        ordenar(pob, true);
        int pos, aux2=0;
        double random;
        pob.getIndividuos().ElementAt(0).setPuntAcum(0, pob.getPuntuacion());

        for (int i = 1; i < pob.getIndividuos().Length;i++ ){
pob.getIndividuos().ElementAt(i).setPuntAcum(pob.getIndividuos().ElementAt(i -
1).getPuntAcum(), pob.getPuntuacion());
        }

        for (int i = 0; i < pob.getIndividuos().Length;i++ ) {
           random = Convert.ToDouble(new Random());
           for (pos = 0; pos < pob.getIndividuos().Length;pos++){
              if(pob.getIndividuos().ElementAt(pos).getPuntAcum() > random || pos ==
pob.getIndividuos().Length-1){
                 aux.Replace(aux2,pob.getIndividuos().ElementAt(pos));
                 aux2++;
              }
           }
        }
        return aux;
    }
```

Torneo

```
public Poblacion torneo(Poblacion pob)
    {
        Poblacion aux = new Poblacion(pob.getIndividuos().Length);
```

```
        int pos1, pos2, pos3, aux2 = 0 ;

        for (int i = 0; i < pob.getIndividuos().Length; )
        {
           pos1 = ((Convert.ToInt32(new Random())*1000)/pob.getIndividuos().Length);
           pos2 = ((Convert.ToInt32(new Random()) * 1000) / pob.getIndividuos().Length);
           pos3 = ((Convert.ToInt32(new Random()) * 1000) / pob.getIndividuos().Length);

           if (pob.getIndividuos().ElementAt(pos1).getEvaluacion() >
          pob.getIndividuos().ElementAt(pos2).getEvaluacion())
           {
              if (pob.getIndividuos().ElementAt(pos1).getEvaluacion() >
              pob.getIndividuos().ElementAt(pos3).getEvaluacion())
              {
                 aux.Replace(aux2, pob.getIndividuos().ElementAt(pos1));
              }
              else
              {
                 aux.Replace(aux2, pob.getIndividuos().ElementAt(pos3));
              }
           }
           else
           {
              if (pob.getIndividuos().ElementAt(pos2).getEvaluacion() >
              pob.getIndividuos().ElementAt(pos3).getEvaluacion())
              {
                 aux.Replace(aux2,pob.getIndividuos().ElementAt(pos2));
              }
              else
              {
                 aux.Replace(aux2, pob.getIndividuos().ElementAt(pos3));
              }
           }
           aux2++;
        }

        return aux;
     }
```

Universal Estocástico

```
public Poblacion universalEstocastico(Poblacion pob)
     {
        Double intervalo = ((Double)1) / (Double)pob.getIndividuos().Length; //Intervalo entre
                                                                              marcas
        Double marcador = Convert.ToDouble(new Random()) ; //Primer marcador
        Double[] puntos = new Double[pob.getIndividuos().Length+1]; //Guardamos puntos
                                                                    para seleccionar
        Poblacion aux = new Poblacion(pob.getIndividuos().Length);
```

```
        int aux2 = 0;
        puntos[0] = (Double)0;
        ordenar(pob, false);

        for (int i = 1; i < pob.getIndividuos().Length;i++ )
        {
           puntos[i] = puntos[i-1] +
           (pob.getIndividuos().ElementAt(i).getEvaluacion()/pob.getPuntuacion());
        }
        puntos[puntos.Length] = (Double)1;

        for (int j = 0; j < pob.getIndividuos().Length; j++)
        {
           for (int i = 1; i < puntos.Length; i++)
           {
              if (marcador >= puntos[i - 1] && marcador < puntos[i])
              {
                 aux.Replace(aux2, pob.getIndividuos().ElementAt(i));
                 i = puntos.Length + 3;
                 aux2++;
              }
           }
           marcador += intervalo;
           if (marcador > 1)
           {
              marcador = 0;
           }
        }

        return aux;
     }
```

Ranking

```
public Poblacion ranking(Poblacion pob)
     {
        Poblacion aux = new Poblacion(pob.getIndividuos().Length);
        Double random;
        int aux2 = 0;
        ordenar(pob, true);

        for (int j = 0; j < pob.getIndividuos().Length;j++ )
        {
           random = Convert.ToDouble(new Random());
           for (int i = 0; i < pob.getIndividuos().Length;i++ )
           {
              if(random <= ((i+1)/pob.getIndividuos().Length))
              {
```

```
                aux.Replace(aux2, pob.getIndividuos().ElementAt(i));
                aux2++;
              }
            }
          }
          return aux;
        }
```

CRUCE

El cruce se basa en la reproducción biológica, es decir que, a partir de dos cromosomas que harán de padres, se mezclarán sus genes para generar nuevos individuos. Hemos implementado dos métodos de cruce de cromosomas, Simple y PMX, que se aplican alternativamente a los cromosomas.

Cruce Simple

Elegidos un par de cromosomas, se divide el array de genes en un punto elegido al azar intercambiándose las partes formando un par nuevo de cromosomas. Una vez hecho esto hay que asegurarse que los cromosomas resultantes tienen todas las paradas, de no ser así, se eliminan las duplicadas y se insertan las que faltan.

```
public void cruceSimple(Cromosoma padre1, Cromosoma padre2){
        int punto_cruce = (int) ((Convert.ToInt32(new
Random())*1000/padre1.getGenes().Length));
        Gen aux;

        Cromosoma p1, p2;
        p1 = padre1.Clonar();
        p2 = padre2.Clonar();

        //Cruzamos
        for (int i = punto_cruce; i < p1.getGenes().Length; i++)
        {
           aux = p1.getGenes().ElementAt(i);
           p1.sustituyeGen(p2.getGenes().ElementAt(punto_cruce), punto_cruce);
           p2.sustituyeGen(aux, punto_cruce);

        }

        //Solucionar si faltan paradas
        solucionarParadas(p1, todasParadas);
        solucionarParadas(p2, todasParadas);

        p1.calcularEvaluacion();
```

```
        p2.calcularEvaluacion();

        if(p1.getEvaluacion() > padre1.getEvaluacion()){
           padre1 = p1;
        }
        if(p2.getEvaluacion() > padre2.getEvaluacion()){
           padre2 = p2;
        }
     }
```

Cruce PMX

En este caso el array de genes de los cromosomas se divide por dos puntos al azar intercambiándose el segmento inicial y el final. También debemos comprobar que el cromosoma posea todas las paradas de la ruta.

```
public void crucePMX(Cromosoma padre1, Cromosoma padre2)
     {
        int punto_cruce = (int)((Convert.ToInt32(new Random()) * 1000 /
                                 padre1.getGenes().Length));
        int punto_cruce2 = (int)((Convert.ToInt32(new Random()) * 1000 /
                                  padre1.getGenes().Length));
        int temp;
        Gen aux;

        if(punto_cruce > punto_cruce2){
           temp = punto_cruce;
           punto_cruce = punto_cruce2;
           punto_cruce2 = temp;
        }

        Cromosoma p1, p2;
        p1 = padre1.Clonar();
        p2 = padre2.Clonar();

        for (int i = 0; i < p1.getGenes().Length;i++ )
        {
           if (i < punto_cruce || i >= punto_cruce2)
           {
              aux = p1.getGenes().ElementAt(i);
              p1.sustituyeGen(p2.getGenes().ElementAt(punto_cruce), punto_cruce);
              p2.sustituyeGen(aux, punto_cruce);
           }
        }

        //Solucionar si faltan paradas
        solucionarParadas(p1, todasParadas);
```

```
        solucionarParadas(p2, todasParadas);

        p1.calcularEvaluacion();
        p2.calcularEvaluacion();

        if (p1.getEvaluacion() > padre1.getEvaluacion())
        {
            padre1 = p1;
        }
        if (p2.getEvaluacion() > padre2.getEvaluacion())
        {
            padre2 = p2;
        }
    }
```

MUTACIÓN

La mutación de los cromosomas que hemos implementado, consiste básicamente en intercambiar dos genes, de un mismo cromosoma, de posición haciendo cambiar el orden de visita. Esto sólo se produce cuando el número que hemos generado aleatoriamente es inferior a la probabilidad de mutación, que hemos situado en un nivel muy bajo, (0.05) ya que un elevado porcentaje de mutaciones hace que se pierda una solución cercana a la óptima, debido a los cambios constantes en el individuo mejor.

```
private Cromosoma mutar(Cromosoma m)
    {
        int pos1 = (int) ((Convert.ToInt32(new Random()) * 1000) / m.getGenes().Length);
        int pos2 = (int) ((Convert.ToInt32(new Random()) * 1000) / m.getGenes().Length);
        Gen aux;

        aux = m.getGenes().ElementAt(pos1);
        m.sustituyeGen(m.getGenes().ElementAt(pos2), pos1);
        m.sustituyeGen(aux, pos2);

        return m;
    }
```

ELITISMO

El elitismo es una técnica que consiste en, antes de aplicar los algoritmos genéticos a la población, se salvan los mejores individuos, en este caso el 1,5% (si se coge una cantidad mayor el algoritmo puede convergir muy rápido a un individuo no óptimo); a

continuación, se aplica el algoritmo de manera normal a toda la población y, para finalizar, se sustituyen los peores individuos (el 1,5%) por los elementos salvados anteriormente. De esta manera se garantiza la supervivencia de los mejores individuos de cada población.

```
public Cromosoma[] cogerElite(Poblacion pob){
        Cromosoma[] elite = new Cromosoma[pob.getIndividuos().Length];
        ordenar(pob, false);
        for (int i = 0; i < Math.Ceiling(pob.getIndividuos().Length * 0.015);i++ ){
            elite[i] = pob.getIndividuos().ElementAt(i).Clonar();
        }
        return elite;
    }
public void ponerElite(Cromosoma[] elite, Poblacion pob){
        ordenar(pob, true);
        for (int i = 0; i < elite.Length;i++ ){
           pob.Replace(i,elite[i]);
        }
    }
  }
```

CONEXIÓN CON EL PROVEEDOR DE MAPAS

La aplicación conecta con el proveedor de mapas, en este caso Google Maps, mediante una URL en la que la información está contenida en ella, tanto para enviarla como para recuperarla.

Dependiendo de qué información se quiera pedir o recuperar, la URL se codifica de diferentes maneras. Se especificarán el zoom, las latitudes y longitudes de la dirección de la que queremos tomar referencia. Esta información se recogerá del mapa en cada momento que esté siendo utilizado y la se procesará para crear la URL donde se hará una petición de información al servicio web. De esta manera, recibiremos de vuelta un XML que procesaremos y donde recuperamos la información para transformarla al plano local.

La siguiente imagen muestra el circuito de conexión entre el proveedor de mapas y la aplicación:

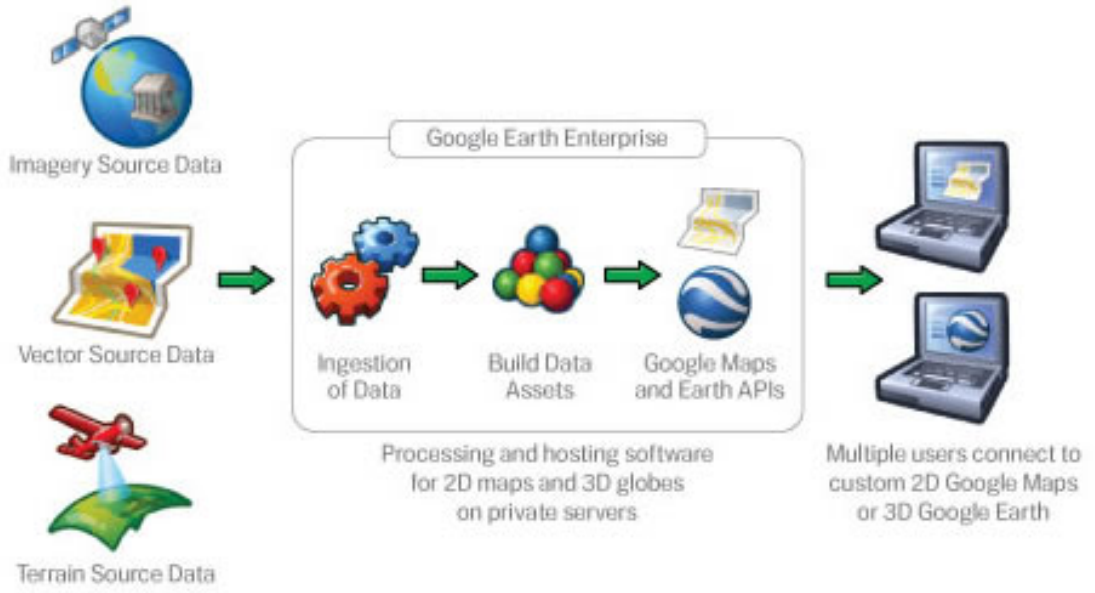

Ilustración 14 - Diagrama de conexión con Google Maps

La construcción de la URL se produce de la siguiente manera: una vez que se tenga una referencia (reference) de una solicitud de búsqueda de sitio, se iniciará una solicitud de detalles de sitio para solicitar más datos sobre un determinado establecimiento o lugar de interés. Una solicitud de detalles de sitio devuelve información más exhaustiva sobre el sitio indicado (por ejemplo, la dirección completa, el número de teléfono y las opiniones y valoraciones de los usuarios).

Una solicitud de detalles de sitio es una URL HTTP con el siguiente formato:

https://maps.googleapis.com/maps/api/place/details/output?parameters

- output: En nuestro caso será “xml”, que generará un archivo de este tipo de donde se procesarán los datos.
- key (obligatorio): indica la clave de API de la aplicación. Esta clave permite identificar la aplicación para la administración de cupos y para que los sitios añadidos desde la aplicación estén disponibles para ella inmediatamente. Para crear un proyecto de API y obtener una clave, accede a la página de la consola de las API.
- reference (obligatorio): indica el identificador textual exclusivo del sitio que se obtiene al enviar una solicitud de búsqueda de sitio.

- sensor (obligatorio): indica si la solicitud de detalles de sitio procede de un dispositivo que utiliza un sensor de ubicación (por ejemplo, un GPS). El valor especificado debe ser true o false.
  El archivo XML que se recibe como respuesta a la URL enviada deberá tiene un formato de este tipo:

```
<?xml version="1.0" encoding="UTF-8"?>
<PlaceDetailsResponse>
 <status>OK</status>
 <result>
  <name>Google Sydney</name>
  <vicinity>48 Pirrama Road, Pyrmont</vicinity>
  <type>establishment</type>
  <formatted_phone_number>(02) 9374 4000</formatted_phone_number>
  <formatted_address>48 Pirrama Road, Pyrmont NSW, Australia</formatted_address>
  <address_component>
   <long_name>48</long_name>
   <short_name>48</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Pirrama Road</long_name>
   <short_name>Pirrama Road</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Pyrmont</long_name>
   <short_name>Pyrmont</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>NSW</long_name>
   <short_name>NSW</short_name>
   <type>administrative_area_level_1</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>AU</long_name>
   <short_name>AU</short_name>
   <type>country</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>2009</long_name>
   <short_name>2009</short_name>
   <type>postal_code</type>
  </address_component>
  <geometry>
   <location>
    <lat>-33.8669710</lat>
    <lng>151.1958750</lng>
   </location>
  </geometry>
  <rating>4.7</rating>
  <url>http://maps.google.com/maps/place?cid=10281119596374313554</url>
  <icon>http://maps.gstatic.com/mapfiles/place_api/icons/generic_business-71.png</icon>
  <reference>CnRsAAAAoGXc0eAcQOoO1A7sU58repRnghwM5q7UtsZFhVtjYtfKN_LFAPhdhBfUAU8
m0EzeSyP0cDBi7kazZwNjlIMUqktqIanMiymuRDS8c539M6KCJNUMkjw22WXxtl3QoR25fIf-
7YJnpza6bMIuFZ1CKBIQuBsbXu8xkbUNofECCkdvmxoU5k3Lpbr8XNCbofIKtsZxj8GloGA</reference
```

```
>
 <id>4f89212bf76dde31f092cfc14d7506555d85b5c7</id>
 <international_phone_number>+61 2 9374 4000</international_phone_number>
 <website>http://www.google.com.au/</website>
 <review>
  <time>1338440552869</time>
  <text>Just went inside to have a look at Google. Amazing.</text>
  <author_name>Simon Bengtsson</author_name>
  <author_url>https://plus.google.com/104675092887960962573</author_url>
  <aspect>
   <type>quality</type>
   <rating>3</rating>
  </aspect>
 </review>
 <review>
  <time>1338411244325</time>
  <text>Best place to work :-)</text>
  <author_name>Felix Rauch Valenti</author_name>
  <author_url>https://plus.google.com/103291556674373289857</author_url>
  <aspect>
   <type>quality</type>
   <rating>3</rating>
  </aspect>
 </review>
 <review>
  <time>1330467089039</time>
  <text>Great place to work, always lots of free food!</text>
  <author_name>Chris</author_name>
  <author_url>https://maps.google.com/maps/user?uid=211457841236072500285</author_url>
  <aspect>
   <type>quality</type>
   <rating>3</rating>
  </aspect>
 </review>
</result>
</PlaceDetailsResponse>
```

Una respuesta XML consta de un único elemento <PlaceDetailsResponse> y tres elementos de nivel superior:

- <status>: contiene los metadatos de la solicitud.
- un elemento <result>: contiene información detallada sobre un establecimiento.
- <html_attributions>: contiene un conjunto de atribuciones que se deben mostrar al usuario.

El campo "status" del objeto de respuesta de sitio contiene el estado de la solicitud y puede incluir información sobre depuración que te ayude a conocer los motivos del fallo de una solicitud de sitio. A continuación, se indican los posibles valores del campo "status".

- OK: indica que no se han producido errores, que se ha detectado correctamente el sitio y que se ha obtenido como mínimo un resultado.
- UNKNOWN_ERROR: indica que se ha producido un error del servidor. El problema se podría resolver intentando volver a enviar la solicitud.

- ZERO_RESULTS: indica que la referencia era válida, pero ya no corresponde a un resultado válido. Esto puede suceder si el establecimiento ya no está operativo.
- OVER_QUERY_LIMIT: indica que se ha superado el límite de solicitudes.
- REQUEST_DENIED: indica que se ha denegado la solicitud, generalmente porque falta un parámetro sensor.
- INVALID_REQUEST: suele indicar que falta la referencia (reference) de la consulta.
- NOT_FOUND: indica que no se ha encontrado la ubicación de referencia en la base de datos de Places.

RESULTADOS DE DETALLES DE DIRECCIONES

Cuando el servicio devuelve resultados de una solicitud de detalles, los incluye en un único resultado (result). A continuación se indican los campos que puede contener cada resultado.

- address_components[]: es un conjunto de componentes de dirección independientes que se utiliza para formar direcciones. Por ejemplo, la dirección "111 8th Avenue, New York, NY" contiene componentes independientes para "111" (el número de la calle), para "8th Avenue" (una ruta), para "New York" (la ciudad) y para "NY" (el estado de EE.UU.). A continuación se indica lo que suele contener cada componente de dirección (address_component).
- types[]: es un conjunto que indica el tipo (*type*) de componente de la dirección.
- long_name: indica la descripción completa o el nombre completo del componente de la dirección que ha devuelto el geocoder.
- short_name: es un nombre textual abreviado del componente de la dirección (si está disponible). Por ejemplo, un componente de la dirección del estado de Alaska puede incluir un nombre completo (long_name) "Alaska" y un nombre abreviado (short_name) "AK" en el que se utilice la abreviación postal de dos letras.
- Un conjunto events[] o uno o varios elementos <event>: proporcionan información sobre eventos que se están desarrollando en ese sitio. Se muestra un máximo de diez eventos (ordenados por hora de inicio). A continuación se indica lo que contiene cada evento.
- event_id: indica el ID de evento único del evento.
- start_time: indica la hora de inicio del evento expresada como tiempo Unix.
- summary: proporciona una descripción textual del evento. Esta propiedad contiene una cadena cuyo contenido no ha sido depurado por el servidor. La aplicación debe estar preparada para impedir o abordar los intentos de amenazas en caso necesario.
- url: indica una URL que dirige a los detalles del evento. Esta propiedad no se obtiene si no se especifica ninguna URL para el evento.
- formatted_address: es una cadena que contiene la dirección interpretable por humanos del sitio. Esta dirección corresponde frecuentemente a la "dirección

postal", que a veces varía de un país a otro. La dirección suele estar formada por uno o varios campos address_component.

- formatted_phone_number: contiene el número de teléfono del sitio en formato local. Por ejemplo, el formato del número de teléfono (formatted_phone_number) de la oficina de Google en Sídney, Australia, es (02) 9374 4000.
- geometry: campo que contiene la información que se indica a continuación.
- location: indica el valor de latitud y longitud del sitio codificado de forma geográfica.
- icon: indica la URL de un icono sugerido que se puede mostrar al usuario en el mapa junto con el resultado.
- id: contiene un identificador estable único del sitio. Este identificador no se puede utilizar para recuperar información sobre el sitio, pero sí para reunir datos sobre ese sitio y para verificar la identidad de un sitio en diferentes búsquedas. Como el ID (id) puede cambiar ocasionalmente, es recomendable comparar el identificador almacenado de un sitio con el identificador obtenido en posteriores solicitudes de detalles de ese sitio y actualizarlo en caso necesario.
- international_phone_number: contiene el número de teléfono del sitio en formato internacional. El formato internacional incluye el código del país precedido del signo "+". Por ejemplo, el formato del número de teléfono (formatted_phone_number) de la oficina de Google en Sídney, Australia, es +61 2 9374 4000.
- name: contiene el nombre interpretable por humanos del resultado obtenido. En el caso de los resultados de establishment, suele corresponder al nombre canónico de la empresa.
- opening_hours: es un campo que puede contener la información que se indica a continuación.
- open_now: es un valor booleano que indica que el sitio está abierto en ese momento.
- periods[]: es un conjunto de períodos de apertura de siete días que comienzan el domingo, dispuestos en orden cronológico. A continuación se indica lo que puede incluir cada período.
- open: incluye un par de objetos de día y hora que indican cuándo está abierto el sitio.
- day: es un número comprendido entre 0 y 6 que corresponde a los días de la semana empezando por el domingo (por ejemplo, 2 corresponde al martes).
- time: puede contener una hora del día en el formato de 24 horas hhmm (con valores comprendidos entre 0000 y 2359). La hora (time) indicada debe corresponder a la zona horaria del sitio.
- close: incluye un par de objetos de día y hora que indican cuándo está cerrado el sitio.
- rating: indica la valoración del sitio (comprendida entre 0,0 y 5,0) basada en las opiniones de los usuarios. Para obtener información sobre valoraciones más detalladas, examina el contenido del grupo aspects.
- reference: contiene un token que se puede utilizar para enviar consultas al servicio de detalles en el futuro. Este token puede ser diferente de la referencia utilizada en

la solicitud enviada al servicio de detalles. Es recomendable que las referencias de sitios almacenadas se actualicen regularmente. Aunque este token identifica el sitio de forma exclusiva, no ocurre lo mismo a la inversa, es decir, un sitio puede tener varios tokens de referencia válidos.

- aspects: contiene un grupo de objetos AspectRating, cada uno de los cuales indica una valoración de un atributo del establecimiento. El primer objeto del grupo se considera el aspecto principal. A continuación se indica la descripción de cada valoración de aspecto (AspectRating).
- type: indica el nombre del aspecto valorado (p. ej., ambiente, servicio, comida, impresión general, etc.).
- rating: indica la valoración del usuario de ese aspecto específico (entre 0 y 3).
- author_name: indica el nombre del usuario que ha enviado la opinión. Las opiniones anónimas se registran como opiniones de "un usuario de Google".
- author_url: indica la URL del perfil de Google+ de los usuarios (si está disponible).
- text: contiene la opinión del usuario. Cuando se valora una ubicación en Google Places, las opiniones de texto se consideran opcionales; por tanto, este campo puede estar vacío.
- time: indica la hora a la que se ha enviado la opinión, expresada como el número de segundos que han transcurrido desde la medianoche del 1 de enero de 1970 (UTC).
- types[]: contiene un conjunto de tipos de funciones que describen el resultado obtenido. Las respuestas XML incluyen varios elementos <type> si se asigna más de un tipo al resultado.
- url: contiene la URL de la página de sitio de Google oficial del establecimiento. Las aplicaciones deben contener un enlace a la página de sitio de Google o insertar esa página en cualquier pantalla que muestre resultados detallados sobre el sitio al usuario.
- utc_offset: contiene el número de minutos de diferencia entre la zona horaria actual del sitio y UTC. Por ejemplo, en el caso de los sitios de Sídney, Australia, sería 660 (+11 horas respecto a UTC) en horario de verano y, en el caso de los sitios de California sería -480 (-8 horas respecto a UTC) en horario de invierno.
- vicinity: indica una dirección simplificada del sitio que incluye el nombre de la calle, el número y la localidad, pero no la provincia (o el estado), el código postal ni el país. Por ejemplo, en el caso de la oficina de Google en Sídney, Australia, el valor de vicinity es 48 Pirrama Road, Pyrmont.
- website: indica el sitio web autorizado del sitio (por ejemplo, la página principal de la empresa).

CAPÍTULO IV: DESARROLLO

INTRODUCCIÓN AL CAPÍTULO

Este capítulo describirá cómo y con qué herramientas hemos desarrollado la aplicación. Vamos a pasar a describir el entorno de desarrollo que se ha usado en este proyecto, el lenguaje usado y los retos afrontados. Seguidamente comentaremos la implementación de partes críticas de la aplicación ilustrándolas con código si es necesario.

Aunque la parte de diseño de los algoritmos principales (Clustering y algoritmo genético) ha sido la parte más costosa de la creación de la propia aplicación, como las hemos plasmado, cobra una parte vital en el desarrollo de este proyecto.

SOBRE VISUAL STUDIO

Microsoft Visual Studio ha sido, tanto a lo largo de la última década como de la presente, el entorno de desarrollo de referencia para todos aquellos desarrolladores de aplicaciones basadas en sistemas operativos Windows (Microsoft Windows, Windows CE o Windows Mobile, entre otros). Su fiabilidad y robustez a la hora de diseñar, programar, desplegar y depurar aplicaciones son, junto con su versatilidad, el punto fuerte de este paquete integrado.

Ilustración 15 - Visual Studio

Visual Studio se pronuncia como uno de los paquetes más versátiles, debido fundamentalmente al amplio soporte que ofrece para numerosos lenguajes de programación, tales como Visual Basic, Visual J#, ASP.NET, C++ o C#. Otros lenguajes como Python y Ruby también son soportados mediante la instalación de módulos y extensiones adicionales. Además, soporta lenguajes de marcado y estilo tales como XML, CSS o HTML.

El paquete contiene un avanzado editor de código y un depurador, así como un diseñador web. Además, cuenta con un completo editor de estructuras de Bases de Datos.

Esta herramienta no sólo se limita a ofrecer un soporte seguro para crear aplicaciones para entornos Windows. Va más allá, permitiendo a los desarrolladores programar sitios y servicios web, así como aplicaciones para móviles. Con este propósito,

Microsoft ha procurado facilitar la tarea de intercomunicar todo este tipo de aplicaciones a través de un framework creado para tal fin en el año 2002, como es .NET.

VISUAL STUDIO 2010

Como consecuencia del énfasis puesto por Microsoft en facilitar el desarrollo e implementación de paquetes web, surge esta última versión del framework, lanzada el 12 de Abril de 2010.

Los puntos fuertes de esta versión son, sin duda alguna, la facilidad de desarrollo y la integración total con Windows 7, así como la simplificación de las tareas relacionadas con el desarrollo de módulos y servicios web.

Se añadió retrocompatibilidad con múltiples versiones anteriores de .NET (incluyendo la actual4.0) y se consiguió agilizar la implementación de servidores web gracias a IntelliSense -una potente herramienta de documentación e inserción de código, pero el punto que verdaderamente hace falta destacar es el soporte que ofrece para poder interactuar de manera satisfactoria con los servidores IIS (Internet Information Service), que permiten publicar servicios web de manera sencilla.

Dado que este proyecto necesita de conexión con Servicios Web, para recoger información de los proveedores mapas, que el entorno de desarrollo permitiese esta conexión de forma fácil e intuitiva, se tornaba más que imperativo. Por ello elegimos Visual Studio 2010, aunque no lo habíamos utilizado nunca, al utilizar un entorno sencillo y parecido a otros que usamos durante la carrera como puede ser Eclipse, no nos tomó mucho tiempo exprimirlo al máximo.

Otro de los punto clave para elegir este entorno fue las librerías que aporta para manipular de forma sencilla archivos Excel, que componen la base de datos del programa.

LENGUAJE C#

¿QUÉ ES C#?

C# es un lenguaje de programación de propósito general, multiparadigma, imperativo, declarativo, fuertemente tipado y orientado a objetos.

Fue desarrollado y estandarizado por Microsoft en el año 2001 como parte de su, por entonces, nueva plataforma de desarrollo ".NET". Su creación vino motivada por la necesidad de disponer de un lenguaje diseñado desde cero y sin elementos heredados de pasadas versiones de C y otros lenguajes, sin elementos innecesarios que dificultase la utilización e interacción con .NET y que resultase lo más sencillo posible para poder aprovechar su versatilidad y potencia.

Este lenguaje combina los mejores elementos de varios de los lenguajes de mayor difusión como C++, Java o Visual Basic. De hecho, su creador Anders Heljsberg fue también el creador de muchos otros lenguajes y entornos como Turbo Pascal, Delphi o Visual J++. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como C++ con la sencillez de lenguajes como Visual Basic, y que además la migración a este lenguaje por los programadores de C/C++/Java sea lo más inmediata posible.

La implementación más conocida de la especificación de este lenguaje es Visual C#, incluida en todos los productos de la gama Visual Studio.

¿POR QUÉ C#?

Escogimos realizar el proyecto en C# ya que es un lenguaje de programación orientado a objetos similar a java, que es el lenguaje que nos es más familiar ya que es el más utilizado en la carrera, y, además, proporciona bibliotecas para el manejo de ficheros Excel, ya que está estandarizado por Microsoft como parte de su plataforma .NET, lo que nos venía a la perfección para el manejo de la base de datos.

.NET FRAMEWORK

.NET es el *framework* (conjunto de tecnologías) desarrollado por Microsoft y lanzado en el año 2002 orientado fundamentalmente a la creación de software para Internet. Proporciona una enorme variedad de librerías y ofrece una interconexión total entre todos los lenguajes de programación soportados por la suite Visual Studio. Además, abstrae a las aplicaciones de las diferentes plataformas de hardware utilizadas entre diferentes redes.

El objetivo básico de este conjunto de herramientas no es otro que el de obtener un entorno específicamente diseñado para el desarrollo y ejecución del software en forma de servicios, que puedan ser hechos públicos y acceder a ellos a través de Internet, de una forma independiente al hardware, software y lenguaje de programación utilizado en el desarrollo o el soporte de dicha aplicación. Este entorno global es lo que en Microsoft denominan Plataforma. NET, y los servicios ya mencionados son los que denominan como servicios web.

El esfuerzo invertido por Microsoft en el fácil y rápido desarrollo de aplicaciones orientadas a la web se tradujo en el diseño del nuevo lenguaje C# ya comentado en el apartado anterior, con el fin de utilizar éste como bandera del ágil desarrollo futuro en .NET.

Para el desarrollo y ejecución de aplicaciones en este nuevo entorno tecnológico, Microsoft proporciona el conjunto de herramientas conocido como .NET *Framework* SDK, que incluye compiladores de lenguajes como C#, Visual Basic.NET, Managed C++ y JScript.NET específicamente diseñados para crear aplicaciones para él.

ABSTRACCIÓN

El corazón del framework .NET es el CLR (Common Language Runtime). Se trata de la máquina virtual de este paquete, y es la responsable de administrar la ejecución de los programas de .NET . Además, realiza una compilación intermedia de todos los lenguajes soportados en un lenguaje de bajo nivel denominado CIL (Common Intermediate Language), que posteriormente se traduce a instrucciones máquina que son ejecutadas por la CPU. Ésta es la forma a través de la cual .NET consigue abstraer las aplicaciones de manera independiente al lenguaje de programación utilizado.

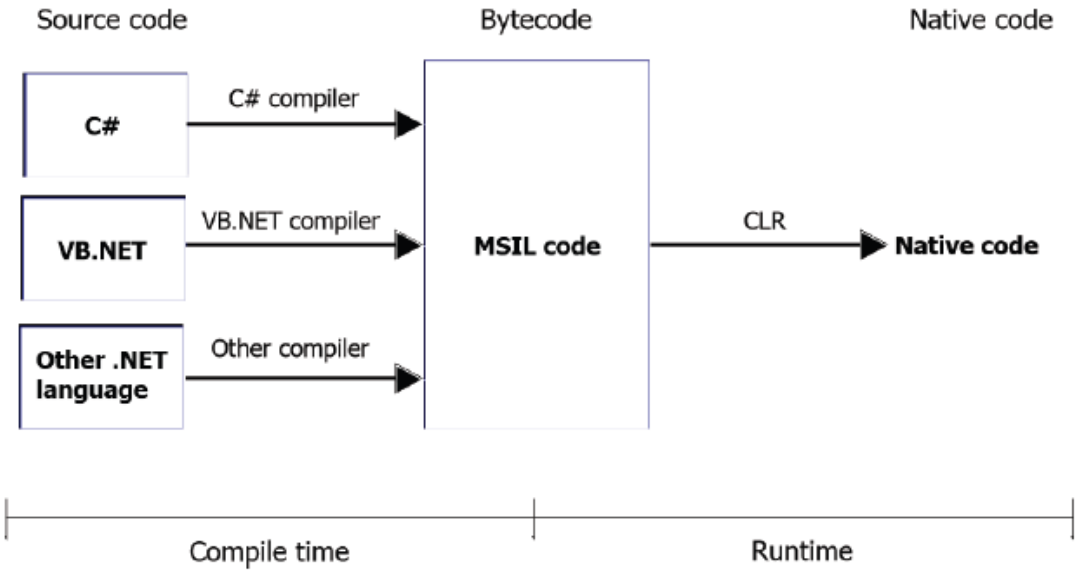

Ilustración 16 - Abstracción de .NET

La última versión estable de .NET en el mercado es la 4.0, lanzada a la par que Visual Studio 2010 (Abril 2010).

MICROSOFT EXCEL

Microsoft Excel es una aplicación distribuida por Microsoft Office para hojas de cálculo. Este programa es desarrollado y distribuido por Microsoft, y es utilizado normalmente en tareas financieras y contables.

Excel ofrece una interfaz de usuario ajustada a las principales características de las hojas de cálculo, en esencia manteniendo ciertas premisas que pueden encontrarse en la hoja de cálculo original, VisiCalc: el programa muestra las celdas organizadas en filas y columnas, y cada celda contiene datos o una fórmula, con referencias relativas, absolutas o mixtas a otras celdas.

Excel fue la primera hoja de cálculo que permite al usuario definir la apariencia (las fuentes, atributos de carácter y celdas). También introdujo recomputación inteligente de celdas, donde celdas dependientes de otra celda que han sido modificadas, se actualizan al instante (programas de hoja de cálculo anterior recalculaban la totalidad de los datos todo el tiempo o esperaban para un comando específico del usuario). Excel tiene una amplia capacidad gráfica, y permite a los usuarios realizar, entre otras muchas aplicaciones, listados usados en combinación de correspondencia.

Excel ha incluido Visual Basic para Aplicaciones (VBA), un lenguaje de programación basado en Visual Basic, que añade la capacidad para automatizar tareas en Excel y para

proporcionar funciones definidas por el usuario para su uso en las hojas de trabajo. VBA es una poderosa anexión a la aplicación que, en versiones posteriores, incluye un completo entorno de desarrollo integrado (IDE) conocido también como Editor de VBA. La grabación de macros puede producir código (VBA) para repetir las acciones del usuario, lo que permite la automatización de simples tareas. (VBA) permite la creación de formularios y controles en la hoja de trabajo para comunicarse con el usuario. Admite el uso del lenguaje (pero no la creación) de las DLL de ActiveX (COM); versiones posteriores añadieron soporte para los módulos de clase permitiendo el uso de técnicas de programación básicas orientadas a objetos.

La aplicación conecta con la base de datos mediante la librería de Visual Studio "Microsoft.Office.Interop", de donde aplicamos los operadores necesarios para manipular los ficheros que componen la base de datos. Creamos las operaciones básicas de gestión de ficheros de datos como puede ser abrir, cerrar, escribir o leer.

Código para abrir el fichero:

```
public void Open(string path)
    {
        this.path = path;
        this.app = new Microsoft.Office.Interop.Excel.Application();
        this.app.Visible = false;
        this.app.ScreenUpdating = false;
        this.app.DisplayAlerts = false;
        this.app.Workbooks.Add();
        this.book = this.app.Workbooks.Open(this.path);
        if (this.book == null)
            throw new Exception("Can't open the excel book file.");
    }
```

Código para leer el fichero:

```
public void Write(string sheet, string cell, string value)
    {
        Worksheet wsheet = this.getSheet(sheet);
        Range = wsheet.get_Range(cell, cell);
        range.Value2 = value;
    }
```

El administrador podrá manipular los datos directamente en la interfaz gráfica de la hoja de cálculo, haciendo el programa sencillo y fácilmente modificable, lo cual lo hace especialmente sencillo para corregir errores.

CAPÍTULO V: CONCLUSIONES

CONCLUSIONES GENERALES

Tras el desarrollo e investigación del proyecto, hemos constatado cómo se debe implementar una herramienta de gestión de rutas de la forma más óptima reduciendo los costes del transporte al mínimo posible. La cualidad diferenciadora de esta aplicación es la combinación de la agrupación en paradas y el algoritmo genético junto al servicio de mapas, que hacen que la representación de la ruta en un escenario real, cobre un gran valor a la hora de obtener benéficos económicos para una empresa de transporte. Muchas empresas de transporte no aplican un estudio de la ruta matemático y, mucho menos aún, basado en nuevas tecnologías como puede ser el web Mapping.

El agrupamiento en paradas aplicando Clustering disminuye mucho el tiempo de la ruta, optimizando el número y la posición de las paradas, variables que no se suelen tener en cuenta en los trayectos.

La motivación para la creación de esta aplicación ha sido dar respuesta a empresas de recogida de pasajeros, pero en general, cualquiera que necesite recorrer varios puntos de forma optimizada y no disponga de una herramienta apta para un usuario con poca experiencia con las nuevas tecnologías. Esto se ha querido conseguir mediante el diseño de una aplicación sencilla y visual, rasgos que concede el uso de bases de datos en Excel y el apoyo en el Servicio de mapas Google Maps.

GUÍA PARA EL USUARIO

El usuario introduce los archivos Excel, con las plantillas que ha enviado cada usuario, en la carpeta que se haya definido para ello.

Posteriormente, se ejecuta el proyecto y, al iniciarse, se recopilaré toda la aplicación de los ficheros Excel, se ejecutan todos los algoritmos de Clustering y los algoritmos genéticos y aparecerá la interfaz que se muestra en la siguiente imagen:

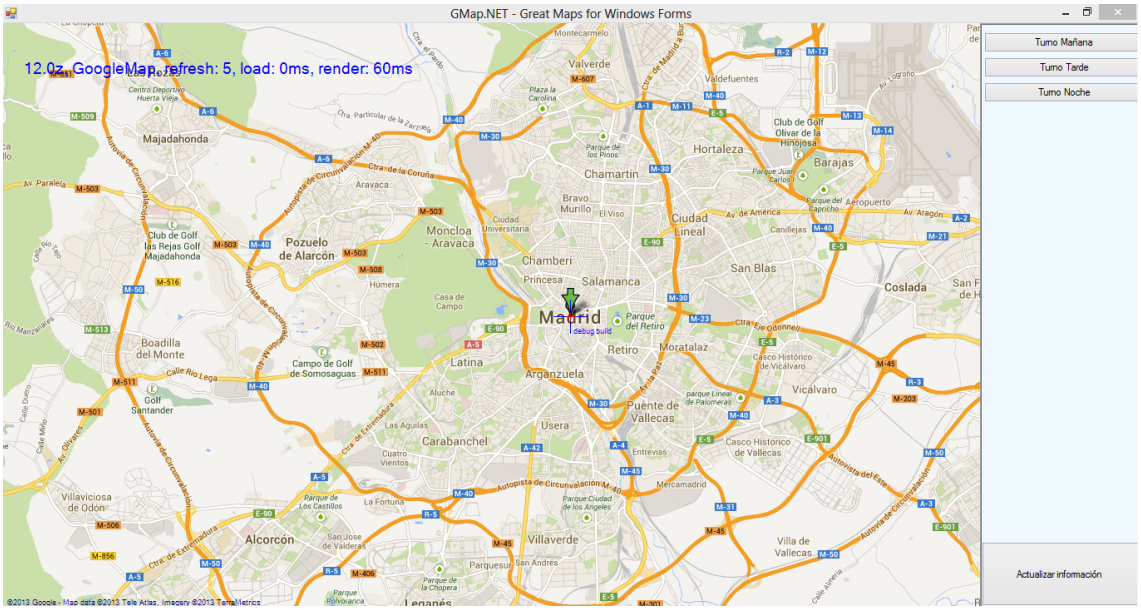

Ilustración 17 - Interfaz de la Aplicación

Esta es la interfaz de la aplicación, sencilla e intuitiva. Cuenta con un visualizador para el mapa que se obtiene a través de Google Maps y un menú lateral en la parte derecha de la ventana. En este menú, encontramos botones para mostrar las rutas en los distintos turnos y para volver a recopilar la información de los archivos Excel por si se han recibido nuevos, no tener que volver a ejecutar la aplicación.

Una vez que hagamos que se muestre una ruta, se mostrará, en el espacio libre del menú lateral, la información de los pasajeros que deben subirse al vehículo en cada parada, como podemos observar en la siguiente imagen:

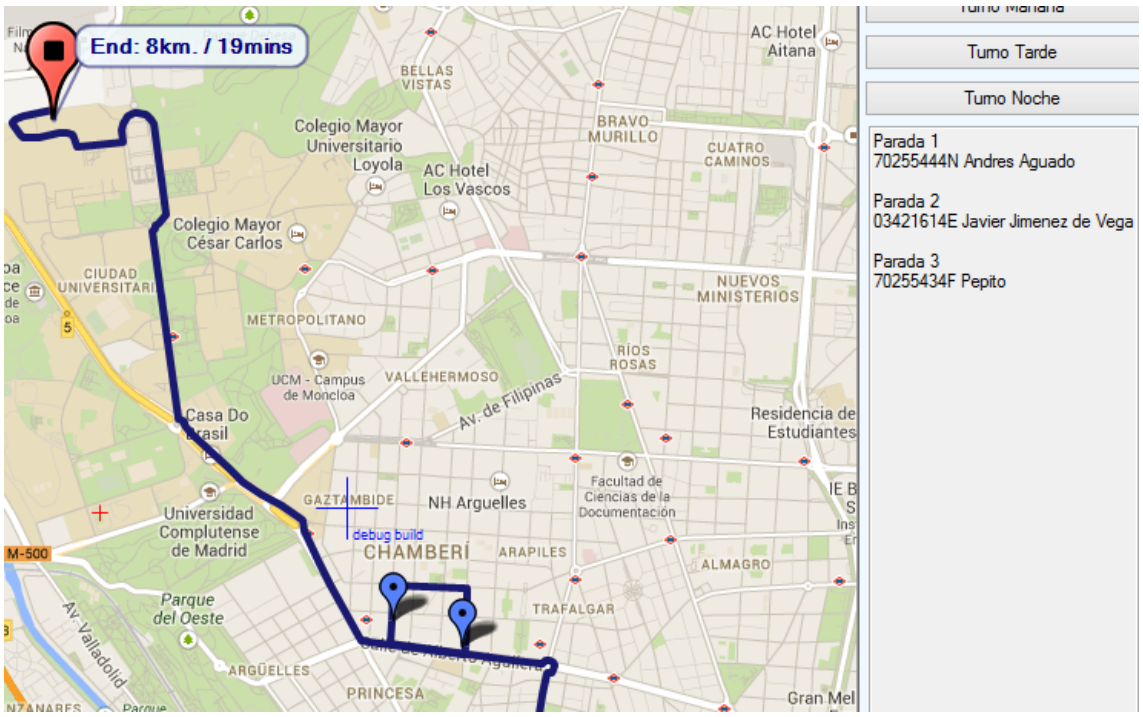

Ilustración 18 - Información de los pasajeros de cada parada

Podemos visualizar la base de datos generada, con la información de las rutas y los pasajeros en la plantilla que está en el mismo lugar donde depositamos los archivos de cada pasajero, con el nombre “BDD”.

RESULTADOS

Durante la realización de este proyecto hemos llegado a diferentes conclusiones en base a los objetivos que planteábamos y las herramientas que hemos utilizado.

Los algoritmos de Clustering ofrecen resultados óptimos para la agrupación de pasajeros. Incluso con altas concentraciones en un perímetro acotado, devuelve agrupaciones de paradas que no hacen sino simplificar la ruta para el recorrido del transporte. El mayor problema que hemos tenido que resolver en este sentido ha sido que durante la carrera no hemos estudiado nada acerca del Clustering lo cual en un principio fue un desafío, pero al final se ha tornado en un concepto nuevo que hemos aprendido y nos puede servir para posteriores aplicaciones.

El algoritmo genético proporciona unos resultados muy coherentes y óptimos para nuestro problema, aunque hemos comprobado que, realmente, según aumenta el número de paradas, los resultados empeoran debido a la codificación del problema.

Esta aplicación ha derivado en una sencilla forma de conocer cómo recorrer distintos puntos de un mapa de la forma óptima. Esto le da gran versatilidad al proyecto, pudiéndose adaptar no solo a la recogida de pasajeros sino a cualquier sistema de transporte logístico, en diversos terrenos.

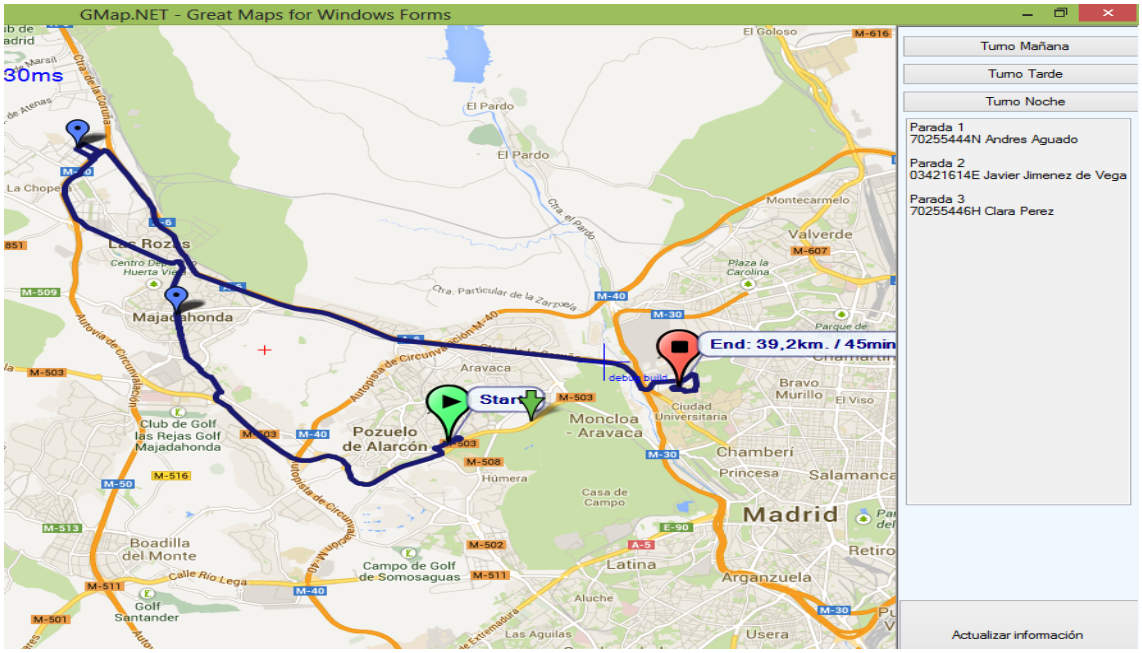

Ilustración 19 -Ejemplo ruta de larga distancia

El resultado general de la aplicación es satisfactorio. Alcanza los objetivos que planteamos en un principio, brindando una aplicación sencilla, tanto para el usuario, como para el administrador, pero muy efectiva a la hora de generar rutas para las empresas para una utilización interactiva de estas.

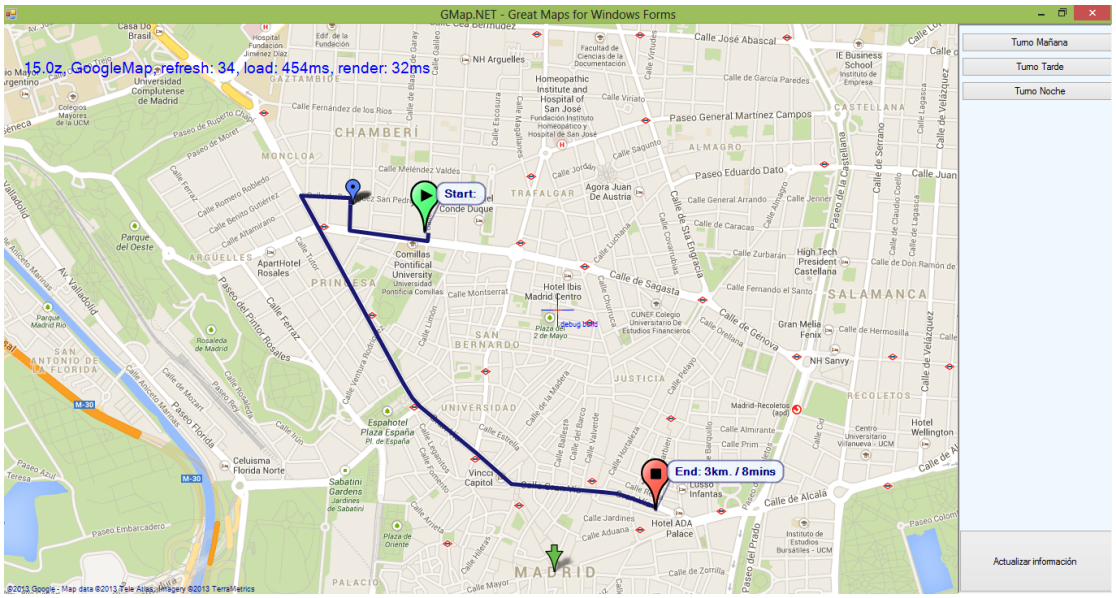

Ilustración 20 - Interfaz mostrando una ruta

POSIBLES AMPLIACIONES

Una de las principales características, que mejoraría en gran medida la funcionalidad de la aplicación, sería crear la aplicación web, para que los usuarios puedan consultar la ruta de transporte. De esta forma, sería accesible desde cualquier punto incluso desde los dispositivos móviles.

Además, se podrían utilizar otras herramientas de optimización como puedes ser el generar un modelo matemático para construir una ruta. Estudiamos esta situación en un principio, pensando en implementarla en el software de optimización de IBM, CPLEX. Esto sustituiría el algoritmo genético, lo cual produciría un resultado seguramente más preciso y en el que podríamos introducir más restricciones a parte del tiempo y la distancia para determinar una solución mejor.

ANEXO: CESIÓN DE DERECHOS

Andrés Aguado Aranda y Francisco Javier Jiménez de Vega, los autores del proyecto y abajo firmantes autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

AGRADECIMIENTOS

A nuestro director de proyecto, José Jaime Ruz Ortiz, por su paciencia y consejo.

A nuestras familias por su apoyo incondicional.

BIBLIOGRAFÍA

- Profesor Carlos Cervigón Ruckauer UCM, Diapositivas de la asignatura de Programación Evolutiva.

- Fernando Berzial Galiano UGR, "Métodos de agrupamiento" Introducción a la minería de datos.

- Brain Trust Consulting Services 2009, "Técnicas para la Optimización de Rutas de Transporte y Distribución".

- Guía para desarrolladores de Google Maps https://developers.google.com/places/documentation/?hl=es

- Microsoft Visual Studio, MSDN, página web <http://msdn.microsoft.com/es-es/vstudio/default.aspx>